Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

А.А. Голубева

ПРОЕКТИРОВАНИЕ И АРХИТЕКТУРА ПРОГРАММНЫХ СИСТЕМ

Методические указания к практическим занятиям и организации самостоятельной работы для студентов направления
«Бизнес-информатика»
(уровень магистратуры)

Томск
2021

УДК 004
ББК 16
Г621

Рецензент:
Сидоров А. А., заведующий кафедрой автоматизации обработки информации Томского государственного университета систем управления и радиоэлектроники, канд. экон. наук, доцент

Голубева Александра Александровна
Проектирование и архитектура программных систем: Методические указания к лабораторным работам и организации самостоятельной работы для студентов направления «Бизнес-информатика» (уровень бакалавриата) / А.А. Голубева. – Томск, 2021. – 77 с.

Методические указания содержат описание лабораторных работ и рекомендации по организации самостоятельной работы студентов в рамках изучения дисциплины «Проектирование и архитектура программных систем» и приобретения практических навыков позволяющих формулировать и решать задачи проектирования информационных систем путем создания функциональных и структурных спецификаций.

Для студентов высших учебных заведений, обучающихся по группе направлений «Экономика и управление».

Одобрено на заседании кафедры АОИ, протокол № 01 от 18.02.2021

УДК 004
ББК 16
Г621

© Голубева А.А., 2021
© Томский государственный университет систем управления и радиоэлектроники, 2021

Оглавление

1 ВВЕДЕНИЕ... 4

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ……………………………………………………………................... 5

2.1 Лабораторная работа «Создание диаграммы прецедентов» 5

2.2 Лабораторная работа «Создание диаграммы классов» 18

2.3 Лабораторная работа «Создание диаграммы состояний и диаграммы деятельности системы» 24

2.4 Лабораторная работа «Создание диаграмм последовательности и коопераций» 43

2.5 Лабораторная работа «Создание диаграмм компонентов и развертывания» 57

3 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ.. 69

4 СПИСОК ЛИТЕРАТУРЫ………………………………… 77

1 ВВЕДЕНИЕ

Целью лабораторных и самостоятельных работ в рамках изучения дисциплины «Проектирование и архитектура программных систем» является формирование у студентов, обучающихся по направлению «Бизнес-информатика», навыков, позволяющих формулировать и решать задачи проектирования информационных систем путем создания функциональных и структурных спецификаций. Кроме того, развивают у обучающихся навыки применения полученных специальных знаний для решения частных задач разработки информационных систем конкретного (специального) назначения.

Лабораторные работы являются важной составляющей в изучении дисциплины и направлены на изучение основных понятий и принципов проектирования информационных систем, ознакомление с современными методиками проектирования. Самостоятельная работа студентов подразумевает под собой различные виды активности, в том числе проработку лекционного материала для подготовки к лабораторным и контрольным работам, выполнение индивидуального задания.

«Приложила исправленный файл. Ваши замечания поправила. Список МУ отправила Коротковой. Прошу проверить и принять работу»

Лабоха Наталия Николаевна вернула на доработку с комментарием «В основной литературе поменяйте у источника ссылку https://biblio-online.ru на https://urait.ru. У источников в основной и дополнительной литературе дважды указан режим доступа. В п. 12.4 добавьте: При изучении дисциплины рекомендуется обращаться к базам данных, информационно-справочным и поисковым системам, к которым у ТУСУРа открыт доступ: https://lib.tusur.ru/ru/resursy/bazy-dannyh»

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

2.1 Лабораторная работа «Создание диаграммы прецедентов»

Цель работы

Лабораторная работа направлена на формирование навыков разработки диаграммы прецедентов. В пособии описаны основные приемы создания, модификации и специфицирования диаграммы прецедентов. Организация материала соответствует последовательности этапов создания модели. В пособии описаны процессы: создания диаграммы прецедентов и задания ее параметров, размещения актеров и прецедентов и задания их спецификаций, установление отношений между элементами диаграммы, добавления и специфицирования прочих элементов. Пособие не содержит описания семантики основных элементов языка UML и приемов объектно-ориентированного анализа и проектирования (ООАП). Эти сведения образуют материал лекционных занятий.

Форма проведения

Выполнение индивидуального задания: в рамках выполнения лабораторной работы студент самостоятельно выбирает любую моделируемую им предметную область.

Форма отчетности

Документы отчетности сдаются на проверку в электронной форме и включают в себя: файл модели MS Visio.

Теоретические основы

Общие сведения

Диаграмма прецедентов (use case diagram) относится к концептуальному представлению системы, описывая назначение системы. Она разрабатывается для достижения следующих целей:

- определить границы и контекст моделируемой системы;
- сформулировать требования к поведению системы;
- создать и зафиксировать исходное концептуальное представление системы с целью его последующей детализации в форме логических и физических моделей;

•подготовить набор артефактов, используемых разработчиками системы для общения с ее заказчиками и будущими пользователями.

Основная идея состоит в представлении системы посредством совокупности прецедентов (use cases) – сервисов, адресованных конкретным потребителям. Любую сущность, взаимодействующую с системой извне, и являющуюся потребителем адресованного ей сервиса называют актером (actor). В роли актера может выступать человек, техническое устройство, программа, или другая система, служащая источником воздействия на моделируемую систему. С внутренней точки зрения прецедент представляет собой совокупность действий, выполняемых системой в ответ на запрос актера и приводящих к значимому для актера результату. При этом соблюдается принцип "черного ящика" – никакой информации о том, каким именно образом реализуется взаимодействие актера и системы, на диаграмме не приводится.

Говоря формально, диаграмма прецедентов представляет собой граф специального вида, основными элементами которого являются прецеденты, актеры и отношения между ними. С целью упрощения восприятия диаграммы и структурирования информации могут применяться пакеты, содержащие как диаграммы, так их элементы.

Прецеденты

Прецеденты предназначены для спецификации общих особенностей поведения моделируемой системы без рассмотрения ее внутренней структуры. Согласно спецификации UML прецеденты обозначаются эллипсом, внутри которого содержится поясняющий текст, называемый именем прецедента. Однако большинство инструментальных сред располагают текст ниже пиктограммы прецедента из соображений удобочитаемости.

Каждый прецедент соответствует отдельному сервису, который моделируемая система предоставляет по запросу пользователя и содержит законченную последовательность действий, приводящую к значимому для потребителя этого сервиса результату.

С системно-аналитической точки зрения прецеденты могут применяться как для выявления внешних требований к проектируемой системе, так и для фиксации поведения уже существующей системы.

Отечественные разработчики дают имена прецедентам, используя либо глагол, либо существительное, обозначающее действие и

поясняющие слова. Поэтому два прецедента, приведенные на рисунке 1 являются эквивалентными.

Рисунок 1 - Эквивалентные прецеденты.

В некоторых русскоязычных изданиях прецеденты называют вариантами использования.

Актеры

Актером является любая внешняя по отношению к моделируемой системе сущность, непосредственно взаимодействующая с ней и использующая ее для достижения определенных целей и решения частных задач. Актеры используются для обозначения согласованного множества ролей, которые могут играть элементы внешней среды при взаимодействии с системой.

Стандартным обозначением актера является пиктограмма приведенная на рисунке 1, под которой располагается имя актера. Инструментальные среды позволяют изменять внешний вид пиктограммы актера, путем наложения на него стереотипов.

Имя актера начинается с заглавной буквы. Часто имена актеров совпадают с должностями, занимаемыми сотрудниками в организации. Например, кассир, продавец, менеджер, руководитель предприятия. Такая ситуация типична для многопользовательских информационных систем, включающих в себя АРМы, ориентированные на использование сотрудниками соответствующих подразделений. Однако, ключевым принципом в формировании имени актера безусловно является роль, которую он играет по отношению к системе. Из этого вытекает необходимость использовать в качестве имени актера нарицательные существительные, а не имена собственные. Во-первых, трудно представить, чтобы кто-либо играл по отношению к системе роль “Иван Грозный”. Вовторых, получаемая в результате модель оказывается не привязанной к персоналиям. Например, сотрудник банка с фамилией Иванова в настоящий момент выступает по отношению к системе в роли операциониста, внося в нее сведения о клиентах банка и состоянии их

счетов. На следующей неделе сотрудник Иванова продвинется по службе и займет пост начальника отдела. Отныне ей будут требоваться сводки и документы итоговой отчетности. Создаваемая диаграмма не должна утратить своей актуальности, поэтому в модель будут введены актеры, имена которых звучат как “Операционист” и “Руководитель отдела”.

Так как актер находится вне системы, его внутренняя структура никак не определяется, принципиально лишь выделять его роль по отношению к системе.

В некоторых русскоязычных изданиях можно встретить перевод термина “actor” как “действующее лицо” или “актор”.

Отношение ассоциации

Язык UML определяет несколько типов отношений для описания взаимодействия различных элементов диаграммы. Основными типами используемых отношений диаграммы прецедентов являются следующие:

- отношение ассоциации (association relationship);
- отношения расширения (extend relationship); отношение включения (include relationship); отношение обобщения (generalization relationship).

Отношение ассоциации – одно из фундаментальных отношений языка UML, используемое в той или иной форме при создании всех канонических диаграмм языка. В общем случае оно обозначает наличие некоторого “знания” у связанных элементов друг о друге. Если ассоциация является направленной, то элемент, от которого отношение исходит, “знает” об элементе к которому отношение направлено.

Чтобы понять семантику отношения ассоциации, возникающего между актером и прецедентом на диаграмме прецедентов, обратимся к формальной модели унифицированного языка моделирования. Введем понятие стереотипа (stereotype), под которым следует понимать расширение словаря UML, позволяющее создавать новые виды строительных блоков (элементов), производные от существующих, но специфичные для конкретной задачи. В языке UML стереотипы используются для выделения категорий элементов модели.

Строго говоря, отношение ассоциации между актером и прецедентом является отношением коммуникации (communication relationship), получаемым из отношения ассоциации путем наложения на него стереотипа “коммуникация” (“communication”). ***Получаемое таким***

образом отношение указывает, какую конкретно роль играет актер при взаимодействии с системой, связывая его с соответствующим прецедентом, а значит, должно было бы иметь вид, приведенный на рис. 2.

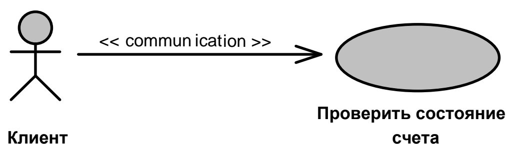

Рисунок 2 - Отношение коммуникации между актером и прецедентом.

Однако, на практике такой подход не получил распространения. Отношения между актером и прецедентом изображается без указания стереотипа, а в различных источниках отношение называют как отношением ассоциации, так и отношением коммуникации. Будем называть отношение между актером и прецедентом отношением ассоциации, но придерживаться более узкого определения, сформулированного выше.

Отношение ассоциации изображается на диаграммах сплошной линией, и дополнительно может характеризоваться направлением, именем и кратностью.

Направление ассоциации призвано показать лицу, читающему диаграмму, кто является инициатором взаимодействия: актер или система. Такая ассоциация называется направленной - unidirectional association. Многие разработчики диаграмм не приводят подобной информации и предпочитают использовать ненаправленные ассоциации (association). Можно сказать, что диаграммы, приведенные на рис. 3 эквиваленты, однако верхняя диаграмма более информативна, поскольку, глядя на нее, можно сказать, что некоторая информационная система предоставляет бухгалтерской системе данные о сотрудниках на основании запроса последней. Инициатором взаимодействия в данном примере является актер.

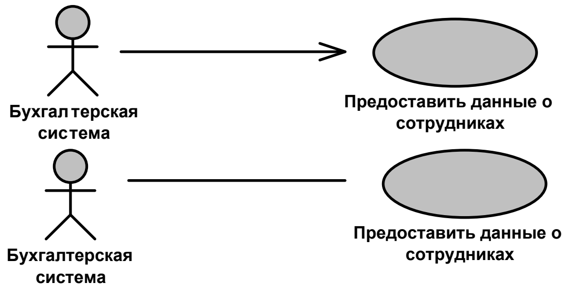

Рисунок 3 - Направленные и ненаправленные ассоциации.

Кратность (multiplicity) ассоциации может быть указана на обоих концах отношения и указывает на количество экземпляров элементов, участвующих в отношении. Рассмотрим диаграмму, приведенную на рис. 4. Некоторый банк занимается оформлением кредитов для своих клиентов. Зафиксируем сказанное при помощи диаграммы прецедентов.

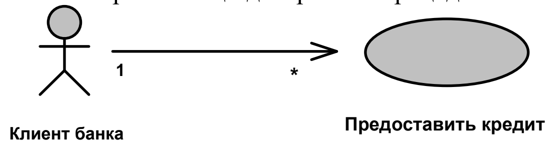

Рисунок 4 - Кратность отношения ассоциации.

Кратность, указанная в форме “*” означает, что каждый отдельный клиент может оформить на себя несколько кредитов. При этом их число не известно, заранее ничем не ограничено и может быть равно 0, т.е. некоторые клиенты могут не брать кредитов вовсе.

На другом конце отношения, кратность указана равной единице. Это означает, что отдельно взятый кредит может оформляться на одного и только одного клиента банка.

Наиболее распространенными формами кратности для отношения ассоциации на диаграмме прецедентов являются следующие.

- Целое неотрицательное число, включая 0. Предназначено для указания кратности, которая является строго фиксированной для участвующего в отношении элемента. Иными словами количество экземпляров элемента участника ассоциации строго равно указанному числу.

- Два целых неотрицательных числа, записанные в форме “первое число .. вторoe число”. Такая запись означает множество целых неотрицательных чисел, следующих в возрастающем порядке с шагом приращения 1. При этом первое число должно быть строго меньше второго в арифметическом смысле и может быть равно 0. В этом случае количество отдельных экземпляров элемента участника ассоциации равно некоторому заранее заданному числу из указанного диапазона целых чисел.

- Два символа, записанные в форме “первое число .. *”. При этом первый символ является целым неотрицательным числом или 0, а второй – специальным символом “*”. Символ “*” означает произвольное конечно целое неотрицательное число, значение которого неизвестно на момент задания отношения ассоциации.

- Единственный символ “*” используется как сокращенная альтернатива записи “0 .. *”. В этом случае количество отдельных экземпляров элемента может быть любым целым неотрицательным числом. При кратности отношения равной 0, отношение ассоциации для данного элемента может вовсе не иметь места.

Если кратность отношения ассоциации не указана, она по умолчанию принимается равной единице.

На практике разработчики диаграмм прибегают к указанию кратности отношения довольно редко. Однако следует помнить, что такой механизм языком унифицированного моделирования предусмотрен и в случае необходимости уметь использовать его для фиксации на диаграмме дополнительной информации.

Возможность давать ассоциациям имена является достаточно удобным механизмом передачи лицу, читающему диаграмму информации о семантике отношения между элементами. Однако, имена ассоциаций на диаграмме прецедентов используют крайне редко, применяя их при создании диаграммы классов.

Отношение расширения

Отношение расширения всегда является направленным и образуется путем наложения стереотипа “extend” (“расширяет”) на отношение зависимости. Его формальное определение выглядит следующим образом. ***Если имеет место отношение расширения, направленное от прецедента А к прецеденту В, это означает, что свойства экземпляра прецедента В могут быть дополнены, благодаря наличию свойств у расширяющего прецедента А.***

Отношение изображается пунктирной стрелкой и направлено от прецедента, расширяющего исходный прецедент, как показано на рис. 5.

<< extend >>

Прецедент А

Прецедент В

Рисунок 5 - Отношение расширения.

Отношение расширения призвано зафиксировать на диаграмме прецедентов тот факт, что один из прецедентов может присоединить к своему поведению некоторое дополнительное поведение, определенное для другого прецедента. Предполагается, что существует некоторое условие и место, называемое точкой расширения, где в поведение базового прецедента включается поведение расширяющего его прецедента, если это условие истинно.

Один прецедент может быть расширяющим для нескольких прецедентов и, в свою очередь, иметь в качестве собственных расширений несколько других прецедентов.

На рисунке 6 приведен пример использования отношения расширения.

<< extend >>

Предоставить

Оформить заказ подробное описание Клиент

Рисунок 6 - Пример использования отношение расширения.

Основным сервисом, который электронный магазин предоставляет клиенту, является оформление заказа. Но, может случиться так, что клиент решит изучить подробные спецификации приобретаемой продукции. Тогда актер запросит у системы сервис подробного описания товара. На изучение подробного описания будет затрачен определенный промежуток времени, на протяжении которого выполнение основного прецедента приостановится.

Для определения точек расширения в форме булевых выражений используются описания потоков событий, которые будут рассмотрены позже.

Отношение включения

Отношение включения всегда является направленным и образуется путем наложения стереотипа “include” (“включает”) на отношение зависимости. Его формально определение выглядит следующим образом. ***Отношение включения, направленное от прецедента А к прецеденту В, указывает, что каждый экземпляр прецедента А включает в себя функциональные свойства прецедента В.***

Отношение изображается пунктирной стрелкой и направлено к прецеденту, включаемому в более общий, как показано на рис. 7.

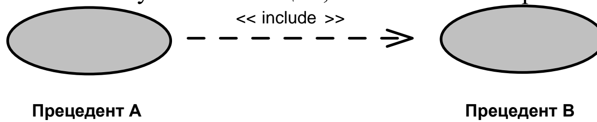

Рисунок 7 - Отношение включения.

Отношения включения между двумя прецедентами указывает, что некоторое поведение одного прецедента включается в качестве составного компонента в последовательность действий другого прецедента, называемого базовым. При этом базовый прецедент может зависеть от результатов выполнения включаемого в него прецедента.

Пример использования отношения включения приведен на рис. 8.

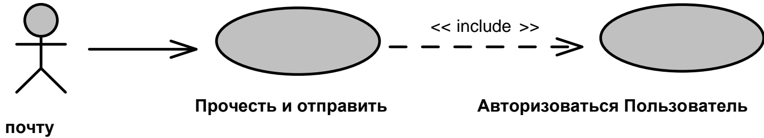

Рисунок 8 - Пример использования отношения включения.

Рассматривая Web-ориентированную систему, позволяющую работать с электронной почтой, основной сервис можно сформулировать как “Чтение и отправка электронной почты”. Однако, доступ к нему возможен лишь после авторизации пользователя в системе, что показано путем введения прецедента “Авторизоваться”, включаемого в базовый прецедент.

Различия между отношениями включения и расширения

Следует остановиться на различиях, существующих между отношениями включения и расширения, изображенные на рис. 9, данные отношения эквивалентны.

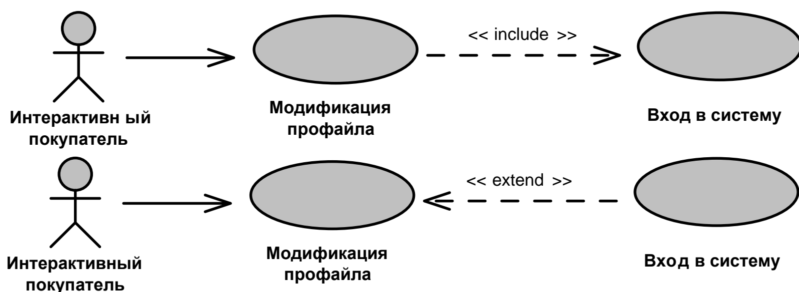

Рисунок 9 - Неэквивалентные диаграммы прецедентов.

Понять разницу между двумя типами отношений довольно легко, если вдумчиво вчитаться в их определение. Отношение расширения “срабатывает” в точке расширения при условии истинности некоторого условия. Т.е. цепочка действий, содержащаяся в прецеденте, расширяющем базовый, может отработать, а может – нет. В отличие от отношения расширения, отношение включения всегда вызывает последовательность действий, содержащуюся в прецеденте, включаемом в базовый. Поэтому если предположить, что в системе, рассматриваемой в работе модификация профайла пользователя невозможна без входа в систему путем авторизации, то справедливой является верхняя диаграмма.

Отношение обобщения

Отношение обобщения (generalization relationship) служит для указания того факта, что некоторый прецедент А может быть обобщен до прецедента В. В этом случае прецедент А будет являться специализацией или потомком прецедента В, а В будет считаться предком или родителем прецедента А. Потомок наследует все свойства и поведение родителя и может быть дополнен новыми свойствами и особенностями поведения.

На диаграмме отношение обобщения представляется сплошной стрелкой, на конце которой располагается не закрашенный треугольник. Отношение обобщения всегда является направленным, указывая на родительский элемент, как показано на рис. 10.

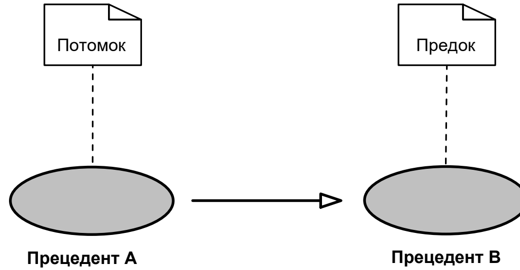

Рисунок 10 - Отношение обобщения между прецедентами.

На практике отношение применяют, когда необходимо показать, что дочерние прецеденты обладают всеми свойствами и особенностями поведения родительских прецедентов. Один дочерний прецедент может быть потомком нескольких родительских. Тогда между прецедентами возникает множественное наследование, и дочерний прецедент наделяется свойствами каждого из прецедентов-предков. Возможна и обратная ситуация – от одного прецедента родителя может быть порождено несколько прецедентов-потомков.

Пример использования отношения обобщения между прецедентами приведен на рис. 11.

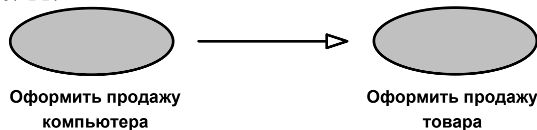

Рисунок 11 - Пример использования отношения обобщения между прецедентами.

В приведенном примере предполагается, что в рамках оформления продажи компьютера выполняется та же последовательность действий, что и при оформлении продажи любого другого товара. Вместе с тем процедура приобретения компьютера сопряжена с определенной спецификой. Например, с оформлением гарантийного талона с перечнем всех комплектующих и указанием их серийных номеров.

Отношение обобщения может возникать и между актерами, как показано на рис. 12. ***Отношение обобщения, направленное от актера А к актеру В призвано отразить тот факт, что каждый экземпляр актера А является одновременно экземпляром актера В и обладает всеми его свойствами.*** В этом случае актер В является предком или родителем по отношению к актеру А, а актер А – его потомком.

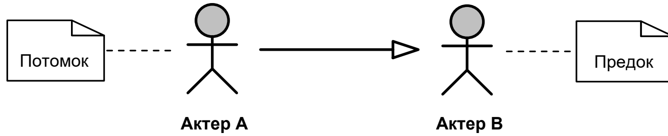

Рисунок 12 - Отношение обобщения между актерами. Пример использования отношения обобщения приведен на рис. 13. Имеется некоторая Web-ориентированная информационная система продажи товаров с использованием сети Internet. Разместить заказ посредством сайта может только зарегистрированный пользователь. Пользователь, не прошедший процедуру регистрации на сайте, может лишь просматривать каталог.

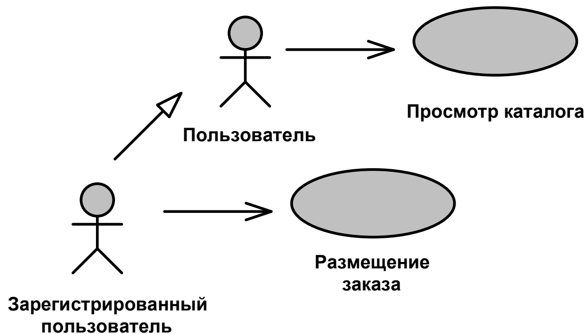

Рисунок 13 - Пример использования отношения обобщения между актерами

Чтобы отразить на диаграмме тот факт, что зарегистрированный пользователь тоже имеет возможность просматривать каталог, введено отношение обобщения между актерами “Пользователь” и “Зарегистрированный пользователь”.

На практике отношение обобщения между прецедентами используется реже, чем отношение обобщения между актерами.

Рекомендации по разработке диаграмм прецедентов

При создании диаграммы прецедентов не следует терять из виду основную цель ее разработки: описать, что будет делать проектируемая система с точки зрения пользователя.

Любой из прецедентов, приведенных на диаграмме, может быть подвергнут дальнейшей декомпозиции. Не следует злоупотреблять этой возможностью и руководствоваться при формулировке прецедентов принципами, изложенными выше. Рекомендуемым решением является переход от абстрактного описания потоков событий прецедентов к более реальному.

В некоторых источниках рекомендуют общее количество актеров, вводимых в модель, не делать большим 20, а прецедентов – большим 50. В противном случае диаграмма прецедентов теряет свою наглядность и, возможно, заменяет собой одну некоторых другим диаграмм.

Если разрабатываемая система является достаточно большой и может быть декомпозирована на подсистемы, целесообразно вводить в модель пакеты, содержащие диаграммы прецедентов и их элементы. При таком подходе иерархия пакетов будет соответствовать иерархии подсистем.

Не моделируйте отношения между актерами. Актеры находятся вне системы, а это означает, что отношения между ними не относятся к сфере моделирования. Исключение составляет отношение обобщения, но и его введение продиктовано ролью актера-предка и актера-потомка по отношению к системе.

Все прецеденты, вводимые в модель, должны быть связаны с актерами: система не должна предоставлять сервисы, которые никем не востребованы. Исключения составляют абстрактные прецеденты, которые не могут иметь экземпляров, а значит, не могут быть запущены. Это правило распространяется и на актеров.

В поиске прецедентов неоценимую роль играет техническое задание и иные документы концептуального характера, создаваемые с привлечением представителей заказчика. Полезно выполнить рассмотрение сферы применения системы на высоком уровне. При этом необходимо учитывать мнение каждого лица, заинтересованного в результатах проекта. Задумайтесь над тем, чего они ожидают от готового продукта.

Порядок выполнения работы

В рамках проведения лабораторной работы необходимо выполнить следующие шаги:

1. Определить моделируемую предметную область.

2. Определить основные сервисы (прецеденты), которые предоставляет система в рамках предметной области.
3. Определить перечень актеров.
4. Определить связи между актерами и прецедентами.
5. Создать диаграмму прецедентов.

2.2 Лабораторная работа «Создание диаграммы классов»

Цель работы

Лабораторная работа направлена на формирование навыков разработки диаграммы классов с использованием инструментальной среды.

Форма проведения

Выполнение индивидуального задания: в рамках выполнения лабораторной работы студент по выбранной в лабораторной работе «Создание диаграммы прецедентов» предметной области создает диаграмму классов.

Форма отчетности

Документы отчетности сдаются на проверку в электронной форме и включают в себя: файл модели MS Visio.

Теоретические основы

Общие сведения

Центральное место в ООАП занимает разработка логической модели системы в виде диаграммы классов. Нотация классов в языке UML проста и интуитивно понятна всем, кто когда-либо имел опыт работы с CASE-инструментариями. Схожая нотация применяется и для объектов — экземпляров класса, с тем различием, что к имени класса добавляется имя объекта и вся надпись подчеркивается.

Нотация UML предоставляет широкие возможности для отображения дополнительной информации: абстрактные операции и классы, стереотипы, общие и частные методы, детализированные интерфейсы, параметризованные классы и т.д.

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектноориентированного программирования. Диаграмма классов отражает различные взаимосвязи между отдельными сущностями предметной области, а также описывает их внутреннюю структуру и типы отношений. На диаграмме классов не указывается информация о временных аспектах функционирования системы.

Диаграмма представляет собой граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами

структурных отношений. Следует заметить, что диаграмма классов может содержать интерфейсы, пакеты, объекты и связи.

Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области.

Класс

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают идентичной 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, как показано на рис. 14. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

Рисунок 14 - Графическое представление класса на диаграмме.

Обязательным элементом обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться прямоугольником с указанием только имени класса (рис. 14, а). По мере проработки отдельных компонентов диаграммы, описания классов дополняются атрибутами (рис. 14, б) и операциями (рис. 14, в).

Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится информация справочного характера или явно указываются исключительные ситуации.

Даже если секция атрибутов и операций является пустой, в обозначении класса она выделяется горизонтальной линией. Примеры графического изображения классов на диаграмме приведены на рис. 15. В первом случае для класса "Прямоугольник" (рис. 15, а) указаны только его атрибуты — точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис. 15, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса "Счет" (рис. 15, в)

дополнительно изображена четвертая секция, в которой указано исключение — отказ от обработки просроченной кредитной карточки.

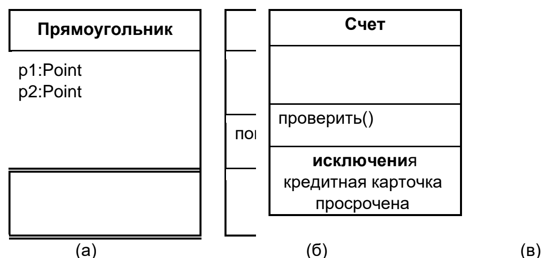

Рисунок 15 - Примеры изображения классов на диаграмме.

Имя класса

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

В некоторых случаях необходимо явно указать, к какому пакету относится тот или иной класс. Для этой цели используется специальный символ разделитель — двойное двоеточие "::". Синтаксис строки имени класса в этом случае будет следующий: ***<Имя_Пакета>::<Имя_Класса>***. Другими словами, перед именем класса должно быть явно указано имя пакета, к которому его следует отнести. Например, если определен пакет с именем "Банк", то класс "Счет" в этом банке может быть записан в виде: "Банк::Счет".

Атрибуты класса

Во второй сверху секции прямоугольника класса записываются его атрибуты 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, типа значений атрибута и, возможно, его исходного значения:

<квантор видимости><имя атрибута> : <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 для всех классов без исключения.

Квантор видимости может быть опущен. В этом случае его отсутствие просто означает, что видимость атрибута не указывается. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

Поскольку язык UML инвариантен относительно реализации своих конструкций в конкретных языках программирования, семантика отдельных кванторов видимости не является строго фиксированной. Значения этих кванторов должны дополнительно уточняться пояснительным текстом на естественном языке или соглашением по использованию соответствующих программно-зависимых синтаксических конструкций.

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и

поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом обозначения атрибута.

Тип атрибута представляет собой выражение, семантика которого определяется языком спецификации соответствующей модели. В нотации UML тип атрибута иногда определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс.

Можно привести следующие примеры задания имен и типов атрибутов классов:

- цвет : Color — здесь цвет является именем атрибута, Color — именем типа данного атрибута. Указанная запись может определять традиционно используемую RGB-модель (красный, зеленый, синий) для представления цвета. В этом случае имя типа Color как раз и характеризует семантическую конструкцию, которая применяется в большинстве языков программирования для представления цвета.
- *видимость : Boolean* — здесь видимость есть имя абстрактного атрибута, который может характеризовать наличие визуального представления соответствующего класса на экране монитора. В этом случае тип Boolean означает, что возможными значениями данного атрибута является одно из двух логических значений: истина (true) или ложь (false). При этом значение истина может соответствовать наличию графического изображения на экране монитора, а значение ложь — его отсутствию. Абстрактный характер данного атрибута обозначается курсивным текстом в записи данного атрибута.
- форма : Многоугольник — здесь имя атрибута форма может характеризовать такой класс, который является геометрической фигурой на плоскости. В этом случае тип атрибута Многоугольник указывает на тот факт, что отдельная геометрическая фигура может иметь форму треугольника, прямоугольника, ромба, пятиугольника и любого другого многоугольника, но не окружности или эллипса. Вполне очевидно, что в данной ситуации использование соответствующего англоязычного термина вряд ли целесообразно, поскольку тип Многоугольник не является базовым для языков программирования.

Исходное значение служит для задания некоторого начального значения для соответствующего атрибута в момент создания отдельного экземпляра класса. Необходимо придерживаться правила принадлежности

значения типу конкретного атрибута. Если исходное значение не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса. С другой стороны, конструктор соответствующего объекта может переопределять исходное значение в процессе выполнения программы, если в этом возникает необходимость.

В качестве примеров исходных значений атрибутов можно привести следующие варианты задания атрибутов:

- цвет : Color = (255, 0, 0) — в RGB-модели цвета это соответствует чистому красному цвету в качестве исходного значения для данного атрибута;
- видимость : Boolean = true — может соответствовать ситуации, когда в момент создания экземпляра класса создается видимое на экране монитора окно, соответствующее данному объекту.
- форма : Многоугольник = прямоугольник — речь идет о геометрической форме создаваемого объекта.

При задании атрибутов могут быть использованы две дополнительные синтаксические конструкции — это подчеркивание строки атрибута и пояснительный текст в фигурных скобках.

Подчеркивание строки атрибута означает, что соответствующий атрибут может принимать подмножество значений из некоторой области значений атрибута, определяемой его типом. Эти значения можно рассматривать как набор однотипных записей или массив, которые в совокупности характеризуют каждый объект класса.

Строка-свойство служит для указания значений атрибута, которые не могут быть изменены в программе при работе с данным типом объектов. Фигурные скобки как раз и обозначают фиксированное значение соответствующего атрибута для класса в целом, которое должны принимать все вновь создаваемые экземпляры класса без исключения. Это значение принимается за исходное значение атрибута, которое не может быть переопределено в последующем. Отсутствие строки-свойства по умолчанию трактуется так, что значение соответствующего атрибута может быть изменено в программе.

Отношения между классами

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

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

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов.

Порядок выполнения работы

В рамках проведения лабораторной работы необходимо выполнить следующие шаги:

1. Определить перечень классов системы.
2. Определить основные атрибуты и методы классов.
3. Определить отношения взаимодействия между классами.
4. Создать диаграмму классов.

2.3 Лабораторная работа «Создание диаграммы состояний и диаграммы деятельности системы»

Цель работы

Лабораторная работа направлена на формирование навыков разработки диаграммы состояний/деятельности системы. Диаграммы состояний/деятельности применяются для документирования динамики поведения объекта, которым может являться сущность предметной области, класс, или система в целом. Как и другие диаграммы языка UML, диаграммы состояний дают команде разработчиков возможность обсудить и документировать логику приложения до начала этапа кодирования.

Форма проведения

Выполнение индивидуального задания: в рамках выполнения лабораторной работы студент по выбранной в лабораторной работе «Создание диаграммы прецедентов» предметной области создает диаграммы состояний и деятельности системы.

Форма отчетности

Документы отчетности сдаются на проверку в электронной форме и включают в себя: файл модели MS Visio.

Теоретические основы

Диаграмма состояний Общие сведения

Любая информационная система характеризуется некоторым поведением и функциональностью. Для общего представления функциональности предназначены диаграммы прецедентов, описывающие на концептуальном уровне сервисы, которые предоставляет актерам моделируемая система. Для того, чтобы выяснить, в процессе какого поведения система реализует эту функциональность в языке UML применяются сразу несколько канонических диаграмм, одной из которых является диаграмма состояний (statechart diagram).

Однако семантика понятия состояния представляет определенные трудности. Поэтому при рассмотрении состояний системы приходится на время отвлечься от особенностей ее объектной структуры и мыслить категориями, образующими динамический контекст поведения моделируемой системы.

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 объекта. При этом изменение состояния объекта может быть вызвано внешними воздействиями со стороны других объектов или извне. В данном случае термин “объект” трактуется шире, чем это принято в ООАП, который определяет его как экземпляр некоторого класса. В роли объекта может выступать система в целом, ее часть, отдельные компоненты и даже актеры.

Главное предназначение диаграммы состояний — 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 системы.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и владеть общими сведениями теории автоматов.

Автоматы

Автомат (state machine) в языке UML представляет собой некоторый формализм, предназначенный для моделирования поведения элементов системы или системы в целом. Автомат описывает поведение 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, срабатывающие при наступлении событий "выход из строя" и "восстановление работоспособности". Графически эта информация может быть представлена в виде изображенной на рис. 16 диаграммы состояний компьютера.

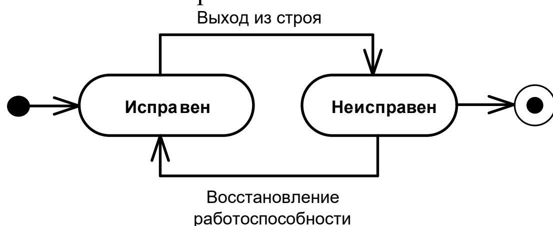

Рисунок 16 - Простейшая диаграмма состояний

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 об этом ничего не сказано). Другими словами, переход объекта из состояния в состояние происходит мгновенно.

В общем случае автомат представляет динамические аспекты моделируемой системы в виде ориентированного графа, вершины которого соответствуют состояниям, а дуги — переходам. При этом поведение моделируется как последовательное перемещение по графу состояний от вершины к вершине по связывающим их дугам с учетом их ориентации.

Для графа состояний системы определены специальные свойства. Одним из таких свойств является выделение из всей совокупности состояний двух специальных: начального и конечного. Хотя на диаграмме состояний время нахождения системы в том или ином состоянии явно не учитывается, предполагается, что последовательность изменения состояний упорядочена во времени. Другими словами, каждое последующее состояние всегда наступает позже предшествующего ему состояния.

Еще одним свойством графа состояний является достижимость состояний. Речь идет о том, что навигация или ориентированный путь в графе состояний определяет специальное бинарное отношение на множестве всех состояний системы. Это отношение характеризует потенциальную возможность перехода системы из рассматриваемого состояния в некоторое другое состояние. Очевидно, для достижимости состояний необходимо наличие связывающего их ориентированного пути в графе состояний.

Формализм автоматов допускает вложение одних автоматов в другие для уточнения внутренней структуры отдельных, более общих состояний, называемых макросостояниями. В этом случае вложенные автоматы называют подавтоматами. Например, состояние неисправности компьютера, приведенное на рис. 16, может быть детализировано на отдельные подсостояния, каждое из которых может характеризовать неисправность отдельных подсистем, входящих в состав данного устройства.

Формализм обычного автомата основан на выполнении следующих обязательных условий:

- Автомат не запоминает историю перемещения из состояния в состояние. С точки зрения моделируемого поведения определяющим является сам факт нахождения объекта в том или ином состоянии, но никак не последовательность состояний, в результате которой объект перешел в

текущее состояние. Другими словами, автомат "забывает" все состояния, которые предшествовали текущему в данный момент времени.

Данное условие может быть изменено явным образом для сохранения некоторых аспектов предыстории поведения объекта на основе введения в рассмотрение так называемых исторических состояний, которые в пособии не рассматриваются.

• В каждый момент времени автомат может находиться в одном и только в одном из своих состояний. Это означает, что формализм автомата предназначен для моделирования последовательного поведения, когда объект в течение своего жизненного цикла последовательно проходит через все свои состояния. При этом автомат может находиться в отдельном состоянии как угодно долго, если не происходит никаких событий.

Это условие ограничивает применение автоматов для моделирования последовательных процессов. Необходимость моделирования параллельных процессов приводит к рассмотрению в контексте одной модели нескольких автоматов, каждый из которых специфицирует отдельный процесс поведения.

• Хотя процесс изменения состояний автомата происходит во времени, явно концепция времени не входит в формализм автомата. Это означает, что длительность нахождения автомата в том или ином состоянии, а также время достижения того или иного состояния никак не специфицируются. Другими словами, время на диаграмме состояний присутствует в неявном виде, хотя для отдельных событий может быть указан интервал времени и в явном виде.

Концепция времени в явной форме учитывается при построении диаграммы деятельности, когда требуется синхронизировать во времени процессы взаимодействия нескольких объектов модели. Поскольку диаграмма состояний предназначена для моделирования поведения объекта, которое определяется асинхронными событиями, эти события могут происходить в заранее неизвестные моменты времени.

• Количество состояний автомата должно быть обязательно конечным (в языке UML рассматриваются только конечные автоматы), и все они должны быть специфицированы явным образом. При этом отдельные начальное и конечное состояния могут не иметь спецификаций. В этом случае их назначение и семантика полностью определяются из контекста модели и рассматриваемой диаграммы состояний.

• Граф автомата не должен содержать изолированных состояний и переходов. Это условие означает, что для каждого из состояний, кроме начального, должно быть определено предшествующее состояние.

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

- 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. Исключением являются параллельные подавтоматы, рассматриваемые ниже. В языке UML исключение конфликтов возможно на основе введения сторожевых условий.

Таким образом, правила поведения объекта, моделируемого некоторым автоматом, определяются общим формализмом автомата и его графическим изображением в языке UML посредством диаграммы состояний.

Состояние

Вся концепция динамической системы основывается на понятии состояния системы. Однако семантика состояния в языке UML имеет целый ряд специфических особенностей. ***В языке UML под состоянием (state) понимается элемент модели, используемый для фиксации отдельной ситуации, в течение которой имеет место выполнение некоторого условия.*** С одной стороны, состояние позволяет зафиксировать ситуацию, когда объект находится в состоянии ожидания возникновения некоторого внешнего события. С другой стороны, состояние используется для моделирования динамических аспектов, когда в период времени, когда оно имеет место, выполняются некоторые действия. В этом случае моделируемый элемент переходит в рассматриваемое состояние в момент начала соответствующей деятельности и покидает данное состояние в момент ее завершения.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами, как показано на рис. 17. Этот прямоугольник может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния. В противном случае в первой из них записывается имя состояния, а во второй — список внутренних действий.

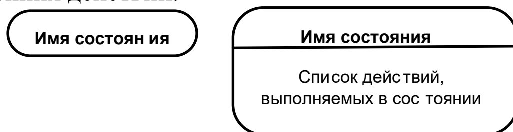

Рисунок 17 -Графическое изображение состояний на диаграмме.

Имя состояния представляет собой строку текста, которая раскрывает содержательный смысл данного состояния. Имя всегда записывается с заглавной буквы. Поскольку состояние системы является составной частью процесса ее функционирования, рекомендуется в качестве имени использовать глаголы в настоящем времени (звенит, печатает, ожидает) или соответствующие причастия (занят, свободен, передано, получено).

Секция списка внутренних действий содержит 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 формат:

<метка действия> '/' <выражение действия>.

Метка действия указывает на обстоятельства или условия, при которых будет выполняться действие, определенное выражением действия. В UML для метки действия имеются три предопределенные значения:

entry — 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 деятельность ("do activity"), которая выполняется в течение всего времени, пока объект находится в данном состоянии.

Разработчик вправе вводить собственные метки действия, придерживаясь следующего правила: метка действия идентифицирует событие, которое запускает расположенное после нее выражение действия.

Начальное и конечное состояния

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. 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 залитой окружности, из которой может только выходить стрелка, соответствующая переходу.

Переход из начального состояния может быть помечен событием создания (инициализации) объекта. В противном случае переход никак не помечается.

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не может содержать 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 залитой окружности, помещенной в окружность большего диаметра. Все переходы для конечного состояния могут быть только входящими.

Переход

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. В этом случае говорят, что переход срабатывает, или происходит срабатывание перехода. До срабатывания перехода объект находится в предыдущем от него состоянии, называемым исходным состоянием, или в состоянии-источнике, а после его срабатывания объект находится в последующем от него состоянии, называемом целевым.

На переходе могут указываться события, вызывающие его срабатывание и действия, производимые объектом 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. При этом 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, а исходное и целевое состояния перехода совпадают. Такой 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 (например, "выход из

строя" на рис. 18). Каждый переход может быть помечен строкой текста, которая имеет следующий общий формат:

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

В свою очередь сигнатура события описывает некоторое событие с необходимыми аргументами: ***<имя события>'('<список параметров>')'.***

Термин ***событие (event)*** требует отдельного пояснения, поскольку является самостоятельным элементом языка UML. 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.

Семантика события фиксирует внимание на внешних проявлениях качественных изменений, происходящих при переходе моделируемого объекта из состояния в состояние. Например, после успешного ремонта компьютера происходит немаловажное событие — восстановление его работоспособности. Если поднять трубку обычного телефона, то, в случае его исправности, мы ожидаем услышать тоновый сигнал. И факт подачи сигнала тоже является событием.

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 на рис. 18 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. После имени события могут следовать круглые скобки для явного задания параметров соответствующего события-триггера. Если таких параметров нет, то список параметров со скобками может отсутствовать.

Сторожевое условие (guard condition), если оно есть, всегда записывается в прямых скобках после события-триггера и представляет собой некоторое булевское выражение.

Введение для перехода сторожевого условия позволяет явно специфицировать семантику его срабатывания. Если сторожевое условие принимает значение "истина", то соответствующий переход может сработать, в результате чего объект перейдет в целевое состояние. 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. Однако вычисление истинности сторожевого условия происходит только после возникновения ассоциированного с ним события-триггера, инициирующего соответствующий переход.

В общем случае из одного состояния может быть несколько переходов с одним и тем же событием-триггером. При этом никакие два сторожевых условия не должны одновременно принимать значение "истина". Каждое из сторожевых условий необходимо вычислять всякий раз при наступлении соответствующего события-триггера.

Примером события-триггера может служить окончание загрузки электронного сообщения 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 разорвать соединение с провайдером, что и делает программа-клиент. В случае отрицательного ответа "ложь", следует оставаться в состоянии загрузки почты и не разрывать телефонное соединение.

Графически фрагмент логики моделирования почтовой программы может быть представлен в виде диаграммы состояний, приведенной на рис. 18.

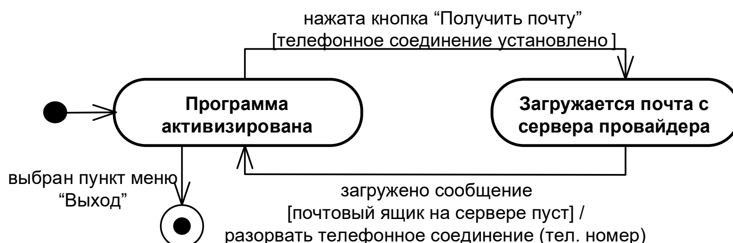

Рисунок 18 - Диаграмма состояний почтовой программы-клиента.

Как можно заключить из контекста, в начальном состоянии программа не выполняется, хотя и имеется на компьютере пользователя. В момент ее запуска происходит активизация. В этом состоянии программа может находиться неопределенно долго, пока пользователь не выгрузит ее из оперативной памяти компьютера, выбрав пункт меню “Выход”. В активном состоянии программы пользователь может читать сообщения электронной почты, создавать собственные и выполнять другие действия, не указанные явно на диаграмме.

Переход в состояние “Загружается почта с сервера провайдера” срабатывает при наступлении события “Нажата кнопка “Получить почту”. При этом пользователь должен установить телефонное соединение с провайдером, что и показано явно на диаграмме в форме сторожевого условия. В противном случае (линия занята, неверный ввод пароля) никакой загрузки почты не произойдет, и программа останется в прежнем своем состоянии.

Выражение действия (action expression) выполняется в том и только в том случае, когда переход срабатывает. Выражение действия 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. Выражение записывается после знака "/" в строке текста, присоединенной к соответствующему переходу. В общем случае, выражение действия может содержать целый список отдельных действий, разделенных символом ";". Обязательное требование — все действия из списка должны четко различаться между собой и следовать в порядке их записи. На синтаксис записи выражений действия не накладывается никаких ограничений. Главное — их запись должна быть понятна разработчикам модели и программистам.
Поэтому часто выражения записывают на одном из языков программирования, который предполагается использовать для реализации модели.

В качестве примера выражения действия, как показано на рис. 2.3.3 может служить "разорвать телефонное соединение (телефонный номер)”,

которое должно быть выполнено сразу после установления истинности сторожевого условия "почтовый ящик на сервере пуст".

Другим примером может служить очевидная ситуация с выделением графических объектов на экране монитора при однократном нажатии левой кнопки мыши. Имеется в виду обработка сигналов от пользователя при выделении тех или иных графических примитивов (пиктограмм). В этом случае соответствующий переход может иметь следующую строку текста: "нажата и отпущена левая кнопка мыши (координаты) [координаты в области графического объекта] / выделить объект (цвет)".

Результатом этого триггерного перехода может быть, например, активизация некоторых свойств объекта или последующее его удаление в корзину.

Составное состояние и подсостояние

Составное состояние (composite state) — сложное состояние, состоящее 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.

Последовательные подсостояния (sequential substates) используются для моделирования такого поведения объекта, во время которого в каждый момент времени объект может находиться в одном и только одном подсостояний. Поведение объекта в этом случае представляет собой последовательную смену подсостояний, начиная от начального и заканчивая конечным подсостояниями. Хотя объект продолжает находиться в составном состоянии, введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты его внутреннего поведения.

Параллельные подсостояния

Параллельные подсостояния (concurrent substates) позволяют специфицировать два и более подавтомата, которые могут выполняться параллельно внутри составного события. Каждый из подавтоматов занимает некоторую область (регион) внутри составного состояния, которая отделяется от остальных горизонтальной пунктирной линией. Если на диаграмме состояний имеется составное состояние с вложенными параллельными подсостояниями, то объект может одновременно находиться в каждом из этих подсостояний.

Однако отдельные параллельные подсостояния могут, в свою очередь, состоять из нескольких последовательных подсостояний. В этом случае по определению объект может находиться только в одном из последовательных подсостояний подавтомата.

Рекомендации

Основные особенности построения диаграмм состояний были рассмотрены при описании соответствующих элементов модели. Сформулируем рекомендации, не нашедшие своего отражения выше.

По своему назначению диаграмма состояний не является обязательным представлением в модели и как бы "присоединяется" к тому элементу, который, по замыслу разработчиков, имеет нетривиальное поведение в течение своего жизненного цикла. Наличие у системы нескольких состояний, отличающихся от простой дихотомии "исправен — неисправен", "активен — неактивен", "ожидание — реакция на внешние действия", может рассматриваться как признак необходимости построения диаграммы состояний.

При выделении состояний и переходов следует помнить, что длительность срабатывания отдельных переходов должна быть существенно меньшей, чем нахождение моделируемого объекта в соответствующих состояниях. Каждое из состояний должно характеризоваться определенной устойчивостью во времени. Другими словами, из каждого состояния на диаграмме не может быть самопроизвольного перехода в какое бы то ни было другое состояние. Все переходы должны быть явно специфицированы, в противном случае построенная диаграмма состояний является либо неполной, либо ошибочной.

При разработке диаграммы состояний нужно постоянно следить, чтобы объект в каждый момент мог находиться только в единственном состоянии. Если это не так, то данное обстоятельство может быть, как следствием ошибки, так и неявным признаком наличия параллельности поведения моделируемого объекта. В последнем случае следует явно специфицировать необходимое число подавтоматов, вложив их в то составное состояние, которое характеризуется нарушением условия одновременности.

Диаграмма деятельности

Общие сведения

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. До появления языка UML для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Например, чтобы позвонить по телефону, предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций.

Важно подчеркнуть то обстоятельство, что с увеличением сложности системы возрастает значение строгого соблюдения последовательности выполняемых операций. Попытка заварить кофе холодной водой приведет к непригодности одной порции напитка. Нарушение последовательности операций при ремонте двигателя может привести к его поломке или выходу из строя. Еще более катастрофические последствия могут произойти в случае отклонения от установленной последовательности действий при взлете или посадке авиалайнера, запуске ракеты, регламентных работах на АЭС.

Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, и переход в следующее состояние срабатывает только при ее завершении. Графически диаграмма деятельности представляется в форме графа, вершинами которого являются состояния действия, а дугами — переходы от одного состояния действия к другому.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Именно они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий. Довольно часто диаграмм деятельности применяются для визуализации особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние на диаграмме сопоставляют факту выполнения операции некоторого класса (либо ее части), используя диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. Отдельные элементарные вычисления, приводящие к некоторому результату, называют действием (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате осуществления деятельности. Результат может привести к изменению состояния системы или возвращению некоторого значения.

Хотя диаграмма деятельности предназначена для моделирования поведения систем, время в явном виде на ней отсутствует. Ситуация во многом аналогична диаграмме состояний.

Состояние действия и состояние деятельности

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.

Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами, как показано на рис. 19. Внутри этой фигуры записывается **выражение действия (action expression)**, которое должно быть уникальным в пределах одной диаграммы деятельности.

Разработать план проекта

N : = N + 1

Рисунок 19 - Графическое изображение состояния действия.

Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами. Если же действие может быть представлено в некотором формальном виде, то целесообразно записать его на том языке программирования, на котором предполагается реализовывать конкретный проект.

Состояния действия не могут быть подвергнуты декомпозиции, поскольку они являются атомарными. Это значит, что в период пребывания системы в состоянии действия выполняемая работа не может быть прервана. Обычно предполагается, что длительность пребывания системы в состоянии действия занимает неощутимо малое время.

В противоположность этому состояния деятельности могут быть подвергнуты дальнейшем декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время. Можно считать, что состояние действия - это частный вид состояния деятельности, а конкретнее - такое состояние, которое не может быть подвергнуто дальнейшей декомпозиции. А состояние деятельности можно представлять себе как составное состояние, поток управления которого включает только другие состояния деятельности и действий. Состояния деятельности и действий обозначаются одинаково, с тем отличием, что у первого могут быть дополнительные части, такие как действия входа и выхода (то есть выполняемые соответственно при входе в состояние и выходе из него), и оно может сопровождаться спецификациями подавтоматов. В этом случае можно использовать специальное обозначение для состояния, содержащего под-деятельности. Такое состояние обозначается специальной пиктограммой в правом нижнем углу символа состояния деятельности, как показано на рис. 20.

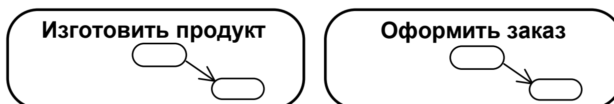

Рисунок 20 - Графическое изображение состояния деятельности, содержащего вложенные действия.

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний. При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии. Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз или слева направо. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное — в ее нижней части.

Переход

Переход как элемент языка UML был рассмотрен при изучении диаграммы состояний. При построении диаграммы деятельности используются только нетриггерные переходы, т. е. такие, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. Этот переход переводит систему в последующее состояние сразу, как только закончится действие в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, то сработать может только один из них. Именно в этом случае для каждого из таких переходов должно быть явно записано сторожевое условие в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата. Такая ситуация получила название ветвления, а для ее обозначения применяется специальный символ.

Ветвление

Графически ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста. В этот ромб может входить только одна стрелка от того состояния действия, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа ветвления. Выходящих стрелок может быть две или более, но для каждой из них явно

указывается соответствующее сторожевое условие в форме булевского выражения.

Разделение и слияние

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений. Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем, необходимы графические примитивы для представления параллельных процессов. В языке UML для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления. Таким символом является прямая черточка, аналогично обозначению перехода в формализме сетей Петри.

Как правило, такая черта изображается отрезком горизонтальной линии, толщина которой несколько шире основных линий диаграммы деятельности. При этом **разделение (concurrent fork)** имеет один входящий переход и несколько исходящих. **Слияние (concurrent join)**, напротив, имеет несколько входящих переходов и один выходящий.

Точка разделения соответствует расщеплению одного потока управления на два, выполняющихся параллельно. В этой точке может существовать ровно один входящий переход и два или более исходящих. Каждый исходящий переход представляет собой один независимый поток управления. После точки разделения деятельности, ассоциированные с каждым путем в графе, продолжают выполняться параллельно. С концептуальной точки зрения имеется в виду истинный параллелизм, то есть одновременное выполнение, но в реальной системе это может как выполняться (если система функционирует на нескольких узлах), так и не выполняться (если система размещена только на одном узле). В последнем случае имеет место последовательное выполнение с переключением между потоками, что дает лишь иллюзию истинного параллелизма.

Точка слияния представляет собой механизм синхронизации нескольких параллельных потоков выполнения. В эту точку входят два ели более перехода, а выходит ровно один. Выше точки слияния деятельности, ассоциированные с приходящими в нее путями, выполняются параллельно. В точке слияния параллельные потоки синхронизируются, то есть каждый из них ждет, пока остальные достигнут этой точки, после чего выполнение продолжается в рамках одного потока.

Обратите внимание на необходимость поддержания баланса между точками разделения и слияния. Число потоков, исходящих из точки разделения, должно быть равно числу потоков, приходящих в соответствующую ей точку слияния.

Рекомендации по построению диаграмм деятельности

Диаграммы деятельности играют важную роль в понимании алгоритмов, заложенных в операции классов, и потоков управления моделируемой системы. Используемые для этой цели традиционные блок-схемы алгоритмов обладают серьёзными ограничениями в представлении параллельных процессов и их синхронизации. Применение дорожек и объектов открывает дополнительные возможности для наглядного представления бизнес-процессов, позволяя специфицировать деятельность подразделений организационных систем.

Содержание диаграммы деятельности во многом напоминает диаграмму состояний, хотя и не тождественно ей. Поэтому многие рекомендации по построению последней оказываются справедливыми применительно к диаграмме деятельности.

На начальных этапах проектирования, когда детали реализации деятельностей в проектируемой системе неизвестны, построение диаграммы деятельности начинают с выделения составных состояний, которые в совокупности дают представление о функционировании системы. В последующем, по мере разработки диаграмм классов и состояний, эти составные состояния деятельности уточняются в виде отдельных вложенных диаграмм деятельности компонентов подсистем, какими выступают классы и объекты.

Вместе с тем отдельные участки рабочего процесса в существующей системе могут быть хорошо отлаженными, и может возникнуть желание зафиксировать алгоритм выполнения действий в проектируемой системе. Тогда строится диаграмма деятельности для этих участков, которая отражает конкретные особенности их выполнения с использованием дорожек и объектов. В последующем такая диаграмма вкладывается в более общие диаграммы деятельности для подсистемы и системы в целом.

Таким образом, процесс объектно-ориентированного анализа и проектирования сложных систем представляется как последовательность итераций нисходящей и восходящей разработки отдельных диаграмм, включая и диаграмму деятельности. Доминирование того или иного из направлений разработки определяется особенностями конкретного проекта и его новизной.

В случае типового проекта большинство деталей реализации действий могут быть известны заранее на основе анализа существующих систем или предшествующего опыта разработки систем-прототипов. Для этой ситуации доминирующим будет восходящий процесс разработки. Использование типовых решений может существенно сократить время разработки и избежать возможных ошибок при реализации проекта.

При разработке проекта новой системы, процесс функционирования которой основан на новых технологических решениях, ситуация представляется более сложной. А именно, до начала работы над проектом могут быть неизвестны не только детали реализации отдельных деятельностей, но и само содержание этих деятельностей становится предметом разработки. В данном случае доминирующим будет нисходящий процесс разработки от более общих схем к уточняющим их диаграммам. При этом достижение такого уровня детализации всех диаграмм, который достаточен для понимания особенностей реализации всех действий и деятельностей, может служить признаком завершения отдельных этапов работы над проектом.

В заключение следует заметить, что диаграмма деятельности, так же как и другие виды канонических диаграмм, не содержит средств выбора оптимальных решений. При разработке сложных проектов проблема выбора оптимальных решений становится весьма актуальной. Рациональное расходование средств, затраченных на разработку и эксплуатацию системы, повышение ее производительности и надежности зачастую определяют конечный результат всего проекта. В такой ситуации можно рекомендовать использование дополнительных средств и методов, ориентированных на аналитико-имитационное исследование моделей системы на этапе разработки ее проекта.

В частности, при построении диаграмм деятельности сложных систем могут быть успешно использованы различные классы сетей Петри (классические, логико-алгебраические, стохастические, нечеткие и др.) и нейронных сетей. Применение этих формализмов позволяет не только получить оптимальную структуру поведения системы на ее модели, но и специфицировать целый ряд дополнительных характеристик системы, которые не могут быть представлены на диаграмме деятельности и других диаграммах UML.

Порядок выполнения работы

В рамках проведения лабораторной работы необходимо выполнить следующие шаги:

1. Определить состояния/деятельности отражающие динамику поведения системы в целом.
2. Определить состояния/деятельности поведения всех объектов системы.
3. Создать диаграммы состояния/деятельности отражающие поведение всех объектов системы и динамику поведения системы в целом.

2.4 Лабораторная работа «Создание диаграмм последовательности и коопераций»

Цель работы

Лабораторная работа направлена на формирование навыков разработки диаграмм последовательности и коопераций, отражающих взаимодействие между собой отдельных элементов системы.

Форма проведения

Выполнение индивидуального задания: в рамках выполнения лабораторной работы студент по выбранной в лабораторной работы «Создание диаграммы прецедентов» предметной области создает диаграммы последовательности и коопераций.

Форма отчетности

Документы отчетности сдаются на проверку в электронной форме и включают в себя: файл модели MS Visio.

Теоретические основы

Диаграммы последовательности

Общие сведения

Одной из характерных особенностей систем различной природы и назначения является взаимодействие между собой отдельных элементов, из которых образованы эти системы. Речь идет о том, что различные составные элементы систем не существуют изолированно, а оказывают определенное влияние друг на друга, что и отличает систему как целостное образование от простой совокупности элементов.

В языке UML взаимодействие элементов рассматривается в информационном аспекте их коммуникации, т. е. взаимодействующие объекты обмениваются между собой некоторой информацией. При этом информация принимает форму законченных сообщений. Другими словами, хотя сообщение и имеет информационное содержание, оно приобретает дополнительное свойство оказывать направленное влияние на своего получателя. Это полностью согласуется с принципами ООАП, когда любые виды информационного взаимодействия между элементами системы должны быть сведены к отправке и приему сообщений между ними.

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Взаимодействия объектов можно рассматривать во времени. Для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Для описания структурных особенностей взаимодействия объектов используется диаграмма кооперации.

Объекты

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения. Одно — слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни, как показано на рис. 21. Внутри прямоугольника записываются имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта, который, как известно, представляет собой экземпляр класса.

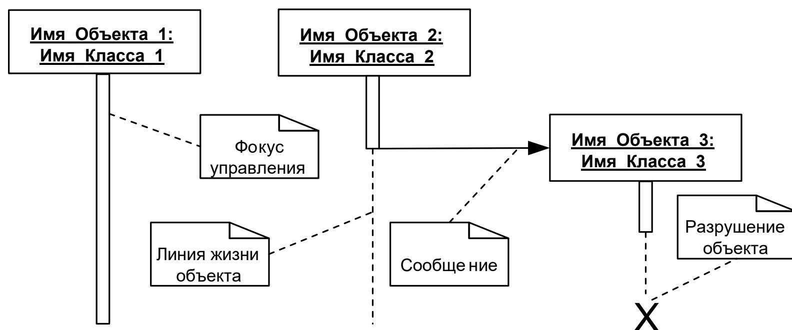

Рисунок 21 - Графические примитивы диаграммы последовательности.

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. На рис. 21 это Объект_1. Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый степенью активности этих объектов при взаимодействии друг с другом.

Второе измерение диаграммы последовательности — вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа "раньше-позже".

Линия жизни объекта

Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его

линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней.

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы "Х". Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее производительность. В этом случае прямоугольник такого объекта изображается не в верхней части диаграммы последовательности, а в той ее части, которая соответствует моменту создания объекта. При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Очевидно, объект обязательно создается со своей линией жизни и, возможно, с фокусом управления.

Фокус управления

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название фокуса управления (focus of control). Фокус управления изображается в форме вытянутого узкого прямоугольника, верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона — окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни, если на всем ее протяжении он является активным.

Периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта будет несколько фокусов управления. Важно понимать, что получить фокус управления может только существующий объект, у которого в этот момент имеется линия жизни. Если же некоторый объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него лишь может быть

создан другой экземпляр этого же класса, который, строго говоря, будет являться другим объектом.

В отдельных случаях инициатором взаимодействия в системе может быть актер или внешний пользователь. В этом случае актер изображается на диаграмме последовательности самым первым объектом слева со своим фокусом управления.

Чаще всего актер и его фокус управления будут существовать в системе постоянно, отмечая характерную для пользователя активность в инициировании взаимодействий с системой. При этом сам актер может иметь собственное имя либо оставаться анонимным.

Иногда некоторый объект может инициировать рекурсивное взаимодействие с самим собой. Речь идет о том, что наличие во многих языках программирования специальных средств построения рекурсивных процедур требует визуализации соответствующих понятий в форме графических примитивов. На диаграмме последовательности рекурсия обозначается небольшим прямоугольником, присоединенным к правой стороне фокуса управления того объекта, для которого изображается это рекурсивное взаимодействие, как у Объекта_4 на рис. 2.4.2.

Сообщения

Цель взаимодействия в контексте языка UML заключается в том, чтобы специфицировать коммуникацию между множеством взаимодействующих объектов. Каждое взаимодействие описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой. В этом смысле сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

Таким образом, сообщения не только передают некоторую информацию, но и требуют или предполагают от принимающего объекта выполнения ожидаемых действий. Сообщения могут инициировать выполнение операций объектом соответствующего класса, а параметры этих операций передаются вместе с сообщением. На диаграмме последовательности все сообщения упорядочены по времени своего возникновения в моделируемой системе.

В таком контексте каждое сообщение имеет направление от объекта, который инициирует и отправляет сообщение, к объекту, который его

получает. 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.

В языке UML могут встречаться несколько разновидностей сообщений, каждое из которых имеет свое графическое изображение, как показано на рис. 22.

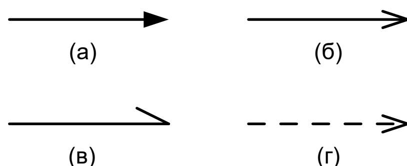

Рисунок 22 - Различные виды сообщений между объектами на диаграмме последовательности.

- Первая разновидность сообщения (рис. 22 а) является наиболее распространенной и используется для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления. Начало этой стрелки всегда соприкасается с фокусом управления или линией жизни того объекта-клиента, который инициирует это сообщение. Конец стрелки соприкасается с линией жизни того объекта, который принимает это сообщение и выполняет в ответ определенные действия. При этом принимающий объект зачастую получает и фокус управления, становясь активным.
- Вторая разновидность сообщения (рис. 22, б) используется для обозначения простого (не вложенного) потока управления. Каждая такая стрелка указывает на прогресс одного шага потока. При этом соответствующие сообщения обычно являются асинхронными, т. е. могут возникать в произвольные моменты времени. Передача такого сообщения обычно сопровождается получением фокуса управления объектом, его принявшим.
- Третья разновидность (рис. 22 в) явно обозначает асинхронное сообщение между двумя объектами в некоторой процедурной последовательности. Примером такого сообщения может служить прерывание операции при возникновении исключительной ситуации. В этом случае информация о такой

ситуации передается вызывающему объекту для продолжения процесса дальнейшего взаимодействия.

- Наконец, последняя разновидность сообщения (рис. 22 г) используется для возврата из вызова процедуры. Примером может служить простое сообщение о завершении некоторых вычислений без предоставления результата расчетов объекту-клиенту. В процедурных потоках управления эта стрелка может быть опущена, поскольку ее наличие неявно предполагается в конце активизации объекта. В то же время считается, что каждый вызов процедуры имеет свою пару — возврат вызова. Для непроцедурных потоков управления, включая параллельные и асинхронные сообщения, стрелка возврата должна указываться явным образом.

Обычно сообщения изображаются горизонтальными стрелками, соединяющими линии жизни или фокусы управления двух объектов на диаграмме последовательности. При этом неявно предполагается, что время передачи сообщения достаточно мало по сравнению с процессами выполнения действий объектами. Считается также, что за время передачи сообщения с соответствующими объектами не может произойти никаких событий. Другими словами, состояния объектов остаются без изменения. Если же это предположение не может быть признано справедливым, то стрелка сообщения изображается под некоторым наклоном, так чтобы конец стрелки располагался ниже ее начала.

В отдельных случаях объект может посылать сообщения самому себе, инициируя так называемые рефлексивные сообщения. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента.

Таким образом, в языке UML каждое сообщение ассоциируется с некоторым действием, которое должно быть выполнено принявшим его объектом. При этом действие может иметь некоторые аргументы или параметры, в зависимости от конкретных значений которых может быть получен различный результат. Соответствующие параметры будет иметь и вызывающее это действие сообщение. Более того, значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвление или альтернативные пути основного потока управления.

Ветвление потока управления

Для изображения ветвления рисуются две или более стрелки, выходящие из одной точки фокуса управления объекта. При этом

соответствующие условия должны быть явно указаны рядом с каждой из стрелок в форме сторожевого условия.

С помощью ветвления можно изобразить сложную логику взаимодействия объектов между собой. Если условий более двух, то для каждого из них необходимо предусмотреть ситуацию единственного выполнения.

Стереотипы сообщений

В языке UML предусмотрены некоторые стандартные действия, выполняемые в ответ на получение соответствующего сообщения. Они могут быть явно указаны на диаграмме последовательности в форме стереотипа рядом с сообщением, к которому они относятся. В этом случае они записываются в кавычках. Используются следующие обозначения для моделирования действий:

- "call" (вызвать) — сообщение, требующее вызова операции или процедуры принимающего объекта. Если сообщение с этим стереотипом рефлексивное, то оно инициирует локальный вызов операции у самого пославшего это сообщение объекта;
- "return" (возвратить) — сообщение, возвращающее значение выполненной операции или процедуры вызвавшему ее объекту. Значение результата может инициировать ветвление потока управления;
- "create" (создать) — сообщение, требующее создания другого объекта для выполнения определенных действий. Созданный объект может получить фокус управления, а может и не получить его;
- "destroy" (уничтожить) — сообщение с явным требованием уничтожить соответствующий объект. Посылается в том случае, когда необходимо прекратить нежелательные действия со стороны существующего в системе объекта, либо когда объект больше не нужен и должен освободить задействованные им системные ресурсы;
- "send" (послать) — обозначает посылку другому объекту некоторого сигнала, который асинхронно инициируется одним объектом и принимается (перехватывается) другим. Отличие сигнала от сообщения заключается в том, что сигнал должен быть явно описан в том классе, объект которого инициирует его передачу.

Кроме стереотипов, сообщения могут иметь собственное обозначение операции, вызов которой они инициируют у принимающего объекта. В этом случае рядом со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все

равно должны присутствовать после имени операции. Примерами таких операций могут служить следующие: "выдать клиенту наличными сумму (n)", "установить соединение между абонентами (a, b)", "сделать вводимый текст невидимым ()", "подать звуковой сигнал тревоги ()".

Согласно принятой в языке UML системе обозначений такие имена операций записываются на английском языке, со строчной буквы и одним словом, возможно, состоящим из нескольких сокращенных слов, написанных без пробела и без кавычек. Если нет никаких дополнительных ограничений со стороны инструментальных средств, то выбор варианта написания имени операции остается за разработчиком. Рекомендуется использовать нижнее подчеркивание, исключающее пробелы в имени операции: "сделать_вводимый_текст_ невидимым()", вместо заглавных буква в середине имени операции: "сделатьВводимыйТекстНевидимым()".

Рекомендации по построению диаграмм

Построение диаграммы последовательности целесообразно начинать с выделения из всей совокупности тех и только тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений. Необходимо установить, какие объекты будут существовать постоянно, а какие временно — только на период выполнения ими требуемых действий.

Когда объекты визуализированы, можно приступать к спецификации сообщений. При этом следует учитывать те роли, которые играют сообщения в системе. При необходимости уточнения этих ролей надо использовать их разновидности и стереотипы. Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение.

Наиболее простые случаи ветвления процесса взаимодействия можно изобразить на одной диаграмме с использованием соответствующих графических примитивов. Однако следует помнить, что каждый альтернативный поток управления может существенно затруднить понимание построенной модели. Поэтому общим правилом является визуализация каждого потока управления на отдельной диаграмме последовательности. В этой ситуации такие отдельные диаграммы должны рассматриваться совместно как одна модель взаимодействия.

Дальнейшая детализация диаграммы последовательности связана с введением временных ограничений на выполнение отдельных действий в

системе. Для простых асинхронных сообщений временные ограничения могут отсутствовать. Однако необходимость синхронизировать сложные потоки управления, как правило, требуют введение в модель таких ограничений. Общая их запись должна следовать семантике языка объектных ограничений, который рассмотрен в приложении.

Диаграмма кооперации

Общие сведения

Особенности взаимодействия элементов моделируемой системы могут быть представлены на диаграммах последовательности и кооперации. Если первая служит для визуализации временных аспектов взаимодействия, то диаграмма кооперации предназначена для спецификации структурных аспектов взаимодействия. Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

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

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

Поведение системы может описываться на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели или реализовать некоторый сервис. С точки зрения

аналитика или конструктора важно представить в проекте системы структурные связи отдельных объектов между собой. Такое статическое представление структуры системы как совокупности взаимодействующих объектов и обеспечивает диаграмма кооперации.

Таким образом, с помощью диаграммы кооперации можно описать полный контекст взаимодействий как своеобразный временной "срез" совокупности объектов, взаимодействующих между собой для выполнения определенной задачи.

Кооперация

Понятие кооперации (collaboration) является одним из фундаментальных понятий в языке UML. Оно служит для обозначения множества взаимодействующих с определенной целью объектов в общем контексте моделируемой системы. Цель самой кооперации состоит в том, чтобы специфицировать особенности реализации отдельных наиболее значимых операций в системе. Кооперация определяет структуру поведения системы в терминах взаимодействия участников этой кооперации.

Кооперация может быть представлена на двух уровнях:

- На уровне спецификации — показывает роли классификаторов и роли ассоциаций в рассматриваемом взаимодействии.
- На уровне примеров — указывает экземпляры и связи, образующие отдельные роли в кооперации.

Диаграмма кооперации уровня спецификации показывает роли, которые играют участвующие во взаимодействии элементы. Элементами кооперации на этом уровне являются классы и ассоциации, которые обозначают отдельные роли классификаторов и ассоциации между участниками кооперации.

Диаграмма кооперации уровня примеров представляется совокупностью объектов (экземплярами классов) и связей (экземплярами ассоциаций). При этом связи дополняются стрелками сообщений. В кооперации уровня примеров определяются свойства, которые должны иметь экземпляры для того, чтобы участвовать в кооперации. Кроме свойств объектов на диаграмме кооперации также указываются ассоциации, которые должны иметь место между объектами кооперации.

Одна и та же совокупность объектов может участвовать в различных кооперациях. При этом, в зависимости от рассматриваемой кооперации, могут изменяться как свойства отдельных объектов, так и связи между ними. Именно это отличает диаграмму кооперации от диаграммы классов,

на которой должны быть указаны все свойства и ассоциации между элементами диаграммы.

В пособии рассматриваются диаграммы кооперации уровня примеров.

Объекты

Отдельные аспекты спецификации объектов как элементов диаграмм уже рассматривались ранее при описании диаграмм последовательности. Поскольку объекты являются основными элементами или графическими примитивами, из которых строится диаграмма кооперации на уровне примеров, рассмотри особенности их спецификации на диаграмме. Для графического изображения объектов используется такой же символ прямоугольника, что и для классов.

Как отмечалось ранее, объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он может иметь свое собственное имя и конкретные значения атрибутов. Применительно к объектам формат строки специфицирования приобретает следующий вид:

<Имя объекта>' / ' <Имя роли класса> ':' <Имя класса> При этом вся запись подчеркивается.

Имя роли класса может не указываться. В этом случае оно исключается из строки текста вместе с последующим двоеточием. Имя роли может быть опущено в том случае, если существует только одна роль в кооперации, которую могут играть объекты, созданные на базе этого класса.

Таким образом, для обозначения роли достаточно указать либо имя класса (вместе с двоеточием), либо имя роли (вместе с наклонной чертой). Если роль, которую должен играть объект, наследуется от нескольких классов, то все они должны быть указаны явно и разделяться запятой и двоеточием.

Связи

Связь (link) является экземпляром ассоциации. Связь как элемент языка UML может иметь место между двумя и более объектами. Бинарная связь на диаграмме кооперации изображается отрезком прямой линии, соединяющей два прямоугольника объектов. На каждом из концов этой линии могут быть явно указаны имена ролей данной ассоциации.

Рядом с линией в ее средней части может записываться имя соответствующей ассоциации.

Связи не имеют собственных имен, поскольку полностью идентичны, будучи экземплярами ассоциации. Другими словами, все связи на диаграмме кооперации могут быть только анонимными и записываются без двоеточия перед именем ассоциации. Для связей не указывается также и кратность. Однако другие обозначения специальных случаев ассоциации (агрегация, композиция) могут присутствовать на отдельных концах связей. Например, символ связи типа "композиция" между мультиобъектом "Принтер" и отдельным объектом "Принтер" на. рис. 8.4.

Связь может иметь некоторые стереотипы, которые записываются рядом с одним из ее концов и указывают на особенность реализации данной связи. В языке UML для этой цели могут использоваться следующие стереотипы:

- "association" — ассоциация (предполагается по умолчанию, поэтому этот стереотип можно не указывать).
- "parameter" — параметр метода. Соответствующий объект может быть только параметром некоторого метода.
- "local" — локальная переменная метода. Ее область видимости ограничена соседним объектом.
- "global" — глобальная переменная. Ее область видимости распространяется на всю диаграмму кооперации.
- "self — рефлексивная связь объекта с самим собой, которая допускает передачу объектом сообщения самому себе. На диаграмме кооперации рефлексивная связь изображается петлей в верхней части прямоугольника объекта.

Сообщения

Сообщения, как элементы языка UML, уже рассматривались ранее при изучении диаграммы последовательности. При построении диаграммы кооперации они имеют некоторые дополнительные семантические особенности. Сообщение на диаграмме кооперации специфицирует коммуникацию между двумя объектами, один из которых передает другому некоторую информацию. При этом первый объект ожидает, что после получения сообщения вторым объектом последует выполнение некоторого действия. Таким образом, именно сообщение является причиной или стимулом для начала выполнения операций, отправки сигналов, создания и уничтожения отдельных объектов. Связь

обеспечивает канал для направленной передачи сообщений между объектами от объектаисточника к объекту-получателю.

Сообщения в языке UML также специфицируют роли, которые играют объекты — отправитель и получатель сообщения. Сообщения на диаграмме кооперации изображаются помеченными стрелками рядом (выше или ниже) с соответствующей связью или ролью ассоциации. Направление стрелки указывает на получателя сообщения. На диаграммах кооперации может использоваться один из четырех типов стрелок для обозначения сообщений, рассмотренных при изучении диаграммы последовательности.

Рекомендации по построению диаграмм кооперации

Построение диаграммы кооперации можно начинать сразу после построения диаграммы классов. Каждая из диаграмм кооперации может уточняться в виде соответствующей диаграммы уровня примеров. Важно понимать, что диаграмма кооперации этого уровня может содержать те и только те объекты и связи, которые уже определены на построенной ранее диаграмме классов. В противном случае, если возникает необходимость включения в диаграмму кооперации объектов, которые создаются на основе отсутствующих классов, соответствующие диаграммы классов должны быть модифицированы явным описанием этих классов.

Следует помнить, что на диаграмме кооперации изображаются только те объекты, которые непосредственно в ней участвуют. При этом объекты могут выступать в различных ролях, которые должны быть явно указаны на соответствующих концах связей диаграммы. Применение стереотипов унифицирует кооперацию, обеспечивая ее адекватную интерпретацию как со стороны заказчиков, так и со стороны разработчиков.

При построении диаграмм кооперации уровня примеров терминология должна наиболее точно отражать все аспекты реализации соответствующих объектов и связей. Поскольку диаграмма этого уровня является документацией для разработчиков системы, здесь допустимо использовать весь арсенал стереотипов, ограничений и помеченных значений, который имеется в языке UML. Если типовых обозначений недостаточно, разработчики могут дополнить диаграмму собственными элементами, используя механизм расширений языка UML.

Процесс построения диаграммы кооперации уровня примеров должен быть согласован с процессами построения диаграммы классов и диаграммы последовательности. В первом случае, как уже отмечалось, необходимо следить за использованием только тех объектов, для которых

определены порождающие их классы. Во втором случае нужно согласовывать последовательности передаваемых сообщений. Речь идет о том, что не допускается различный порядок следования сообщений для моделирования одного и того же взаимодействия на диаграмме кооперации и диаграмме последовательности. Таким образом, диаграмма кооперации, с одной стороны, обеспечивает концептуально согласованный переход от статической модели диаграммы классов к динамическим моделям поведения, представляемым диаграммами последовательности, состояний и деятельности.

Порядок выполнения работы

В рамках проведения лабораторной работы необходимо выполнить следующие шаги:

1. Определить основные элементы системы, для которых необходимо отразить взаимосвязь.
2. Определить типы связей (сообщений между элементами системы).
3. Создать диаграммы последовательности и коопераций.

2.5 Лабораторная работа «Создание диаграмм компонентов и развертывания»

Цель работы

Лабораторная работа направлена на формирование навыков разработки диаграммы компонентов и развертывания системы. Диаграммы компонентов и развертывания служат для физическое представление программной системы, которое не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована.

Форма проведения

Выполнение индивидуального задания: в рамках выполнения лабораторной работы студент по выбранной в лабораторной работе «Создание диаграммы прецедентов» предметной области создает диаграмм компонентов и развертывания.

Форма отчетности

Документы отчетности сдаются на проверку в электронной форме и включают в себя: файл модели MS Visio Studio.

Теоретические основы

Диаграмма компонентов

Общие сведения

Рассмотренные ранее диаграммы отражали концептуальные аспекты построения модели системы и относились к логическому уровню представления. Особенность логического представления заключается в том, что оно оперирует понятиями, которые не имеют материального воплощения. Другими словами, различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Однако для создания конкретной физической системы необходимо некоторым образом реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких

реальных сущностей предназначен другой аспект модельного представления, а именно физическое представление модели.

Чтобы пояснить отличие логического и физического представлений, рассмотрим в общих чертах процесс разработки некоторой программной системы. Ее исходным логическим представлением могут служить структурные схемы алгоритмов и процедур, описания интерфейсов и концептуальные схемы баз данных. Однако для реализации этой системы необходимо разработать исходный текст программы на некотором языке программирования (C++, Pascal, Basic/VBA, Java и др.). При этом уже в тексте программы предполагается такая организация программного кода, которая предполагает его разбиение на отдельные модули.

Тем не менее исходные тексты программы еще не являются окончательной реализацией проекта, хотя и служат фрагментом его физического представления. Очевидно, программная система может считаться реализованной в том случае, когда она будет способна выполнять функции своего целевого предназначения. А это возможно, только если программный код системы будет реализован в форме исполняемых модулей, библиотек классов и процедур, стандартных графических интерфейсов, файлах баз данных. Именно эти компоненты являются необходимыми элементами физического представления системы.

Таким образом, полный проект программной системы представляет собой совокупность моделей логического и физического представлений, которые должны быть согласованы между собой. В языке UML для физического представления моделей систем используются так называемые диаграммы реализации (implementation diagrams), которые включают в себя две отдельные канонические диаграммы: диаграмму компонентов и диаграмму развертывания.

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:
Визуализации общей структуры исходного кода программной системы.

- Спецификации исполнимого варианта программной системы.
- Обеспечения многократного использования отдельных фрагментов программного кода.
- Представления концептуальной и физической схем баз данных.

В разработке диаграмм компонентов участвуют как системные аналитики и архитекторы, так и программисты. Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие — на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами.

Применительно к системам организационного типа программные компоненты следует понимать в более широком смысле, чтобы иметь возможность моделировать бизнес-процессы. В этом случае в качестве компонентов рассматриваются отдельные организационные подразделения (отделы, службы) или документы, которые реально существуют в системе.

Компоненты

Для представления физических сущностей в языке UML применяется специальный термин — компонент (component). Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели. Для графического представления компонента может использоваться специальный символ — прямоугольник со вставленными слева двумя более мелкими прямоугольниками, как показано на рис. 23. Внутри объемлющего прямоугольника записывается имя компонента и, возможно, некоторая дополнительная информация. Изображение этого символа может незначительно варьироваться в зависимости от характера ассоциируемой с компонентом информации.

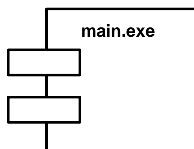

Рисунок 23 - Графическое изображение компонента.

Изображение компонента ведет свое происхождение от обозначения модуля программы, применявшегося некоторое время для отображения особенностей инкапсуляции данных и процедур. Так, верхний маленький прямоугольник концептуально ассоциируется с данными, которые реализует этот компонент (ранее он изображался в форме овала). Нижний маленький прямоугольник ассоциируется с операциями или методами, реализуемыми компонентом. В простых случаях имена данных и методов записывались явно в этих маленьких прямоугольниках, однако в языке UML они не указываются.

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и некоторых знаков препинания. Компонент может быть представлен на уровне типа или на уровне экземпляра. Хотя его графическое изображение в обоих случаях одинаковое, правила записи имени компонента несколько отличаются. Если компонент представляется на уровне типа, то в качестве его имени записывается только имя типа с заглавной буквы.

Если же компонент представляется на уровне экземпляра, то в качестве его имени записывается <имя компонента ':' имя типа> При этом вся строка имени подчеркивается.

Хотя правила именования объектов в языке UML требуют подчеркивания имени отдельных экземпляров, применительно к компонентам в литературе подчеркивание их имени часто опускают. В этом случае запись имени компонента со строчной буквы будет характеризовать компонент уровня экземпляра.

В качестве простых имен принято использовать имена исполняемых файлов (с указанием расширения exe после точки-разделителя), имена динамических библиотек (расширение dll),имена Web-страниц (расширение html), имена текстовых файлов (расширения txt или doc) или файлов справки (hlp),имена файлов баз данных (DB) или имена файлов с исходными текстами программ (расширения h, cpp для языка C++, расширение Java для языка Java), скрипты (pi, asp) и др.

Поскольку конкретная реализация логического представления модели системы зависит от используемого программного инструментария, то и имена компонентов будут определяться особенностями синтаксиса соответствующего языка программирования.

В отдельных случаях к простому имени компонента может быть добавлена информация об имени объемлющего пакета и о конкретной

версии реализации данного компонента. Необходимо заметить, что в этом случае номер версии записывается как помеченное значение в фигурных скобках. В других случаях символ компонента может быть разделен на секции, чтобы явно указать имена реализованных в нем интерфейсов. Такое обозначение компонента называется расширенным и рассматривается ниже.

Виды компонентов

Поскольку компонент как элемент физической реализации модели представляет отдельный модуль кода, иногда его комментируют с указанием дополнительных графических символов, иллюстрирующих конкретные особенности его реализации. Строго говоря, эти дополнительные обозначения для примечаний не специфицированы в языке UML. Однако их применение упрощает понимание диаграммы компонентов, существенно повышая наглядность физического представления. Некоторые из таких общепринятых обозначений для компонентов изображены на рис.
24.

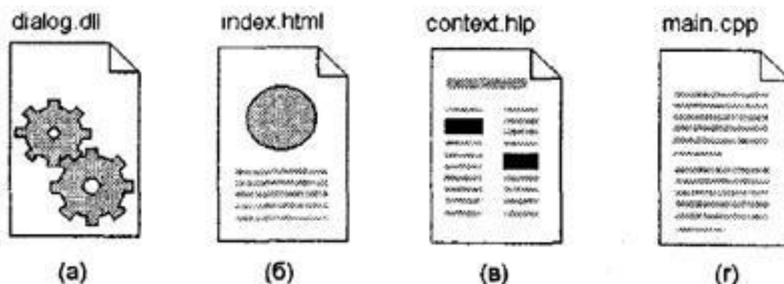

Рисунок 24 - Варианты графического изображения компонентов.

В языке UML выделяют три вида компонентов.

- Во-первых, компоненты развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки с расширением dll (рис. 24 а), Web-страницы на языке разметки гипертекста с расширением html (рис. 24 б) и файлы справки с расширением hlp (рис. 24 в).

- Во-вторых, компоненты-рабочие продукты. Как правило — это файлы с исходными текстами программ, например, с расширениями h или cpp для языка C++ (рис. 24 г).

• В-третьих, компоненты исполнения, представляющие исполнимые модули — файлы с расширением exe. Они обозначаются обычным образом.

Эти элементы иногда называют артефактами, подчеркивая при этом их законченное информационное содержание, зависящее от конкретной технологии реализации соответствующих компонентов. Более того, разработчики могут для этой цели использовать самостоятельные обозначения, поскольку в языке UML нет строгой нотации для графического представления примечаний.

Другой способ спецификации различных видов компонентов — явное указание стереотипа компонента перед его именем. В языке UML для компонентов определены следующие стереотипы:

• Библиотека (library) — определяет первую разновидность компонента, который представляется в форме динамической или статической библиотеки.

• Таблица (table) — также определяет первую разновидность компонента, который представляется в форме таблицы базы данных.

• Файл (file) — определяет вторую разновидность компонента, который представляется в виде файлов с исходными текстами программ.

• Документ (document) — определяет вторую разновидность компонента, . который представляется в форме документа.

• Исполнимый (executable) — определяет третий вид компонента, который может исполняться в узле.

Рекомендации по построению диаграммы компонентов

Разработка диаграммы компонентов предполагает использование информации как о логическом представлении модели системы, так и об особенностях ее физической реализации. До начала разработки необходимо принять решения о выборе вычислительных платформ и операционных систем, на которых предполагается реализовывать систему, а также о выборе конкретных баз данных и языков программирования.

После этого можно приступать к общей структуризации диаграммы компонентов. В первую очередь, необходимо решить, из каких физических частей (файлов) будет состоять программная система. На этом этапе следует обратить внимание на такую реализацию системы, которая обеспечивала бы не только возможность повторного использования кода за счет рациональной декомпозиции компонентов, но и создание объектов только при их необходимости. Речь идет о том, что общая производительность программной системы существенно зависит от

рационального использования ею вычислительных ресурсов. Для этой цели необходимо большую часть описаний классов, их операций и методов вынести в динамические библиотеки, оставив в исполняемых компонентах только самые необходимые для инициализации программы фрагменты программного кода.

После общей структуризации физического представления системы необходимо дополнить модель интерфейсами и схемами базы данных. При разработке интерфейсов следует обращать внимание на согласование (стыковку) различных частей программной системы. Включение в модель схемы базы данных предполагает спецификацию отдельных таблиц и установление информационных связей между таблицами.

Наконец, завершающий этап построения диаграммы компонентов связан с установлением и нанесением на диаграмму взаимосвязей между компонентами, а также отношений реализации. Эти отношения должны иллюстрировать все важнейшие аспекты физической реализации системы, начиная с особенностей компиляции исходных текстов программ и заканчивая исполнением отдельных частей программы на этапе ее выполнения. Для этой цели можно использовать различные виды графического изображения компонентов.

При разработке диаграммы компонентов следует придерживаться общих принципов создания моделей на языке UML. В частности, в первую очередь необходимо использовать уже имеющиеся в языке UML компоненты и стереотипы. Для большинства типовых проектов этого набора элементов может оказаться достаточно для представления компонентов и зависимостей между ними.

Если же проект содержит некоторые физические элементы, описание которых отсутствует в языке UML, то следует воспользоваться механизмом расширения. В частности, использовать дополнительные стереотипы для отдельных нетиповых компонентов или помеченные значения для уточнения их отдельных характеристик.

В заключение следует обратить внимание, что диаграмма компонентов, как правило, разрабатывается совместно с диаграммой развертывания, на которой представляется информация о физическом размещении компонентов программной системы по ее отдельным узлам.

Диаграмма развертывания

Общие сведения

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на

каких вычислительных средствах она реализована. Конечно, если разрабатывается простая программа, которая может выполняться локально на компьютере пользователя, не задействуя никаких периферийных устройств и ресурсов, нет необходимости в разработке дополнительных диаграмм. Однако при разработке корпоративных приложений ситуация представляется совсем по-другому.

Во-первых, сложные программные системы могут реализовываться в сетевом варианте на различных вычислительных платформах и технологиях доступа к распределенным базам данных. Наличие локальной корпоративной сети требует решения целого комплекса дополнительных задач по рациональному размещению компонентов по узлам этой сети, что определяет общую производительность программной системы.

Во-вторых, интеграция программной системы с Интернетом определяет необходимость решения дополнительных вопросов при проектировании системы, таких как обеспечение безопасности, криптозащищенности и устойчивости доступа к информации для корпоративных клиентов. Эти аспекты в немалой степени зависят от реализации проекта в форме физически существующих узлов системы, таких как серверы, рабочие станции, брандмауэры, каналы связи и хранилища данных.

Наконец, технологии доступа и манипулирования данными в рамках общей схемы "клиент-сервер" также требуют размещения больших баз данных в различных сегментах корпоративной сети, их резервного копирования, архивирования, кэширования для обеспечения необходимой производительности системы в целом. Эти аспекты также требуют визуального представления с целью спецификации программных и технологических особенностей реализации распределенных архитектур.

Как было отмечено, первой из диаграмм физического представления является диаграмма компонентов. Второй формой физического представления программной системы является диаграмма развертывания (или диаграмма размещения). Она применяется для представления общей конфигурации и топологии распределенной информационной системы и содержит распределение компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений — маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только

компонентыэкземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, является последним этапом спецификации модели.

Цели, преследуемые при разработке диаграммы развертывания:

- Определить распределение компонентов системы по ее физическим узлам.
- Показать физические связи между всеми узлами реализации системы на этапе ее исполнения.
- Выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Для обеспечения этих требований диаграмма развертывания разрабатывается совместно системными аналитиками, сетевыми инженерами и системотехниками.

Узлы

Узел (node) представляет собой некоторый физически существующий элемент системы, обладающий некоторым вычислительным ресурсом. В качестве вычислительного ресурса узла может рассматриваться наличие по меньшей мере некоторого объема электронной или магнитооптической памяти и/или процессора. В последних версиях UML понятие узла расширено и может включать в себя не только вычислительные устройства (процессоры), но и другие механические или электронные устройства, такие как датчики, принтеры, модемы, цифровые камеры, сканеры и манипуляторы.

Графически на диаграмме развертывания узел изображается в форме трехмерного куба. Узел имеет собственное имя, которое указывается внутри этого графического символа. Сами узлы могут представляться как в качестве типов (рис. 25 а), так и в качестве экземпляров (рис. 25 б).

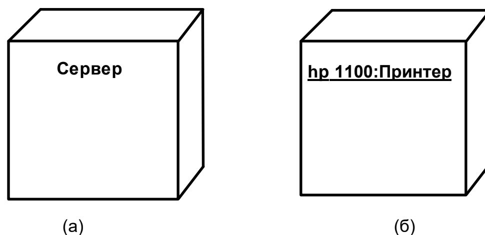

Рисунок 25 - Графическое изображение узла на диаграмме развертывания.

В первом случае имя узла записывается без подчеркивания и начинается с заглавной буквы. Во втором имя узла-экземпляра записывается в виде <имя узла ':' имя типа узла>. Имя типа узла указывает на некоторую разновидность узлов, присутствующих в модели системы.

Так же, как и на диаграмме компонентов, изображения узлов могут расширяться, чтобы включить некоторую дополнительную информацию о спецификации узла. Если дополнительная информация относится к имени узла, то она записывается под этим именем в форме помеченного значения.

Если необходимо явно указать компоненты, которые размещаются на отдельном узле, то это можно сделать двумя способами. Первый из них позволяет разделить графический символ узла на две секции горизонтальной линией. В верхней секции записывают имя узла, а в нижней секции — размещенные на этом узле компоненты.

Второй способ разрешает показывать на диаграмме развертывания узлы с вложенными изображениями компонентов. Важно помнить, что в качестве таких вложенных компонентов могут выступать только исполняемые компоненты.

В качестве дополнения к имени узла могут использоваться различные стереотипы, которые явно специфицируют назначение этого узла. Хотя в языке UML стереотипы для узлов не определены, в литературе встречаются следующие их варианты: "процессор", "датчик", "модем", "сеть", "консоль" и др., которые самостоятельно могут быть определены разработчиком. Более того, на диаграммах развертывания допускаются специальные обозначения для различных физических устройств, графическое изображение которых проясняет назначение или выполняемые устройством функции.

Соединения

Кроме собственно изображений узлов на диаграмме развертывания указываются отношения между ними. В качестве отношений выступают физические соединения между узлами и зависимости между узлами и компонентами, изображения которых тоже могут присутствовать на диаграммах развертывания.

Соединения являются разновидностью ассоциации и изображаются отрезками линий без стрелок. Наличие такой линии указывает на необходимость организации физического канала для обмена информацией между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, помеченным значением или ограничением.

Кроме соединений на диаграмме развертывания могут присутствовать отношения зависимости между узлом и развернутыми на нем компонентами. Подобный способ является альтернативой вложенному изображению компонентов внутри символа узла, что не всегда удобно, поскольку делает этот символ излишне объемным. Поэтому при большом количестве развернутых на узле компонентов соответствующую информацию можно представить в форме отношения зависимости.

Диаграммы развертывания могут иметь более сложную структуру, включающую вложенные компоненты, интерфейсы и другие аппаратные устройства.

Рекомендации по построению диаграммы

Разработка диаграммы развертывания начинается с идентификации всех аппаратных, механических и других типов устройств, которые необходимы для выполнения системой всех своих функций. В первую очередь специфицируются вычислительные узлы системы, обладающие памятью и/или процессором. При этом используются имеющиеся в языке UML стереотипы, а в случае отсутствия последних, разработчики могут определить новые стереотипы. Отдельные требования к составу аппаратных средств могут быть заданы в форме ограничений, свойств и помеченных значений.

Дальнейшее построение диаграммы развертывания связано с размещением всех исполняемых компонентов диаграммы по узлам системы. Если отдельные исполняемые компоненты оказались не размещенными, то подобная ситуация должна быть исключена введением в модель дополнительных узлов, содержащих процессор и память.

При разработке простых программ, которые исполняются локально на одном компьютере, так же как и в случае диаграммы компонентов, необходимость в диаграмме развертывания отсутствует. В более сложных ситуациях диаграмма развертывания строится для таких приложений, как:

- Моделирование программных систем, реализующих технологию доступа к данным "клиент-сервер". Для подобных систем характерно четкое разделение полномочий и, соответственно, компонентов между клиентскими рабочими станциями и сервером базы данных. Возможность реализации "тонких" клиентов на простых терминалах или организация доступа к хранилищам данных приводит к необходимости уточнения не только топологии системы, но и ее компонентного состава.
- Моделирование неоднородных распределенных архитектур. Речь идет о корпоративных интрасетях, насчитывающих сотни компьютеров и других периферийных устройств, функционирующих на различных платформах и под различными операционными системами. При этом отдельные узлы такой системы могут быть удалены друг от друга на сотни километров (филиалы компаний). В этом случае диаграмма развертывания становится важным инструментом визуализации общей топологии системы и контроля миграции отдельных компонентов между узлами.
- Наконец, диаграммы развертывания применимы для моделирования систем со встроенными микропроцессорами, которые могут функционировать автономно. Такие системы могут содержать самые разнообразные дополнительные устройства, обеспечивающие автономность их функционирования и решения целевых задач.

Как правило, разработка диаграммы развертывания осуществляется на завершающем этапе ООАП, что характеризует окончание фазы проектирования физического представления. С другой стороны, диаграмма развертывания может строиться для анализа существующей системы с целью ее последующего анализа и модификации. При этом анализ предполагает разработку этой диаграммы на его начальных этапах, что характеризует общее направление анализа от физического представления к логическому.

Порядок выполнения работы

В рамках проведения лабораторной работы необходимо выполнить следующие шаги:

1. Определить основные компоненты и узлы системы.

2. Определить взаимосвязи между компонентами и узлами системы.

3. Создать диаграммы компонентов и развертывания системы.

3 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

3.1 Общие положения

Целями самостоятельной работы являются проработка тем теоретической части дисциплины вынесенных на самостоятельное изучение, подготовка к контрольным работам и выполнение индивидуального задания.

3.2 Подготовка к контрольным работам

Для успешных выполнений контрольных работ рекомендуется детально проработать лекционный материал по темам:

- Принципы проектирования программных систем.
- Методологии проектирования программных систем.
- Архитектурные стили и модели.

Во время подготовки к контрольным работам необходимо отработать прослушанную лекцию (прочитать конспект, познакомиться с дополнительной литературой) и восполнить пробелы в знаниях, если таковые обнаружились.

Перед каждой последующей контрольной работой повторно прочитать конспект по предыдущим лекциям, чтобы обновить знания.

Примеры контрольных заданий:

1. Принципы проектирования программных систем. Принцип идентичности.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

2. Принципы проектирования программных систем. Принцип технологичности.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

3. Принципы проектирования программных систем. Принцип непрерывности, поэтапности, преемственности разработки и развития.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

4. Принципы проектирования программных систем. Принцип адаптивноси.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

5. Принципы проектирования программных систем. Модульный принцип построения программных и технических средств.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

6. Принципы проектирования программных систем. Технологическая интеграция.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

7. Принципы проектирования программных систем. Полная нормализация процессов и их мониторинг.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

8. Принципы проектирования программных систем. Регламентация.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

9. Принципы проектирования программных систем. Экономическая целесообразность.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

10. Принципы проектирования программных систем. Типизация или максимальное использование готовых решений и средств.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

11. Принципы проектирования программных систем. Стандартизация проектных решений.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

12. Принципы проектирования программных систем. Принцип корпоративности.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

13. Принципы проектирования программных систем. Ориентация на первых лиц объекта автоматизации.

В рамках выполнения контрольного задания необходимо описать принцип. Привести пример.

14. Методологии проектирования программных систем. Scrum. В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения методологии. Привести примеры.

15. Методологии проектирования программных систем. Kanban.

В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения методологии. Привести примеры.

16. Методологии проектирования программных систем. Dynamic system development method.

В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения методологии. Привести примеры.

17. Методологии проектирования программных систем. Microsoft solutions framework.

В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения методологии. Привести примеры.

18. Методологии проектирования программных систем. Rational unified process.

В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения методологии. Привести примеры.

19. Архитектурные стили и модели. Клиент-серверная архитектура. В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения архитектуры. Привести примеры.

20. Архитектурные стили и модели. Компонентная архитектура. В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения архитектуры. Привести примеры.

21. Архитектурные стили и модели. Проблемно-ориентированное проектирование архитектура.

В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения архитектуры. Привести примеры.

22. Архитектурные стили и модели. Многослойная архитектура. В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения архитектуры. Привести примеры.

23. Архитектурные стили и модели. Архитектура на основе канала сообщений.

В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения архитектуры. Привести примеры.

24. Архитектурные стили и модели. Трехуровневая архитектура. В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения архитектуры. Привести примеры.

25. Архитектурные стили и модели. Объектно-ориентированная архитектура.

В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения архитектуры. Привести примеры.

26. Архитектурные стили и модели. Сервисно-ориентированная архитектура.

В рамках выполнения контрольного задания необходимо описать достоинства и недостатки применения архитектуры. Привести примеры.

3.3 Индивидуальное задание «Разработка технического проекта программной системы»

Цель индивидуального задания

Целью индивидуального задания является получение практических и теоретических навыков по разработке технического проекта программной системы.

Порядок выполнения и содержание работ

1. Определиться с содержанием технического проекта программной системы.
2. Произвести выработку и анализ требований к системе.
3. Произвести проектирование системы.
4. Сформировать технический проект программной системы.

3.4 Изучение тем теоретической части дисциплины

Изучение тем теоретической части дисциплины, вынесенных на самостоятельное изучение:

- принципы проектирования программных систем; -
методологии проектирования программных систем;
- архитектурные стили и модели.

В рамках проработки теоретической части дисциплины необходимо:

1) отработать прослушанные лекции (прочитать конспект, просмотреть презентационный материал) и восполнить пробелы в знаниях, если таковые обнаружились;

2) перед каждой последующей лекцией повторно прочитать конспект по предыдущей, чтобы обновить знания для восприятия последующей – новой – информации.

Тема «принципы проектирования программных систем»

Рассмотреть следующие вопросы:

1. Принцип идентичности.
2. Принцип технологичности.
3. Принцип непрерывности, поэтапности, преемственности разработки и развития.
4. Принцип адаптивноси.
5. Модульный принцип построения программных и технических средств.
6. Технологическая интеграция.
7. Полная нормализация процессов и их мониторинг.
8. Регламентация.
9. Экономическая целесообразность.
10. Типизация или максимальное использование готовых решений и средств.
11. Стандартизация проектных решений.
12. Принцип корпоративности.
13. Ориентация на первых лиц объекта автоматизации.

Тема «методологии проектирования программных систем»

Рассмотреть следующие вопросы:

1. Scrum.
2. Kanban.
3. Dynamic system development method.
4. Microsoft solutions framework.
5. Rational unified process.

Тема «архитектурные стили и модели» Рассмотреть следующие вопросы:

1. Клиент-серверная архитектура.
2. Компонентная архитектура.
3. Проблемно-ориентированное проектирование архитектура.
4. Многослойная архитектура.

5. Архитектура на основе канала сообщений.
6. Трехуровневая архитектура.
7. Объектно-ориентированная архитектура. 8. Сервисно-ориентированная архитектура.

4. СПИСОК ЛИТЕРАТУРЫ

1. Золотов, С. Ю. Проектирование информационных систем: Учебное пособие [Электронный ресурс] / Золотов С. Ю. — Томск: ТУСУР, 2016. — 117 с. — Режим доступа: https://edu.tusur.ru/publications/6478.

2. Леоненков А.В. Самоучитель UML / А. В. Леоненков. - 2-е изд. - СПб. : БХВ-Петербург, 2006. - 427 с. В библиотеке ТУСУРа: 20 экз.