7468

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени В.Ф.УТКИНА

UML-МОДЕЛИРОВАНИЕ ФУНКЦИОНАЛЬНОСТИ И ПОВЕДЕНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ

Методические указания
к лабораторным работам
и практическим занятиям

Рязань 2022

УДК 004.942

UML-моделирование функциональности и поведения программно-информационных систем: методические указания к лабораторным работам и практическим занятиям / Рязан. гос. радиотехн. ун-т. им. В.Ф.Уткина; Сост.: В.В. Белов, В.И. Чистякова. Рязань, 2022. 48 с.

Излагаются методические рекомендации по выполнению лабораторных работ и практических заданий по дисциплинам «Проектирование программных систем» и «Проектирование информационных систем». Теоретическую платформу указаний образуют объектно-ориентированная методология проектирования и систематический подход к проектированию, основанный на использовании каскадной модели жизненного цикла программно-информационной системы, принципа ФИСАП и компонентно-ориентированного программирования.

Предназначены для бакалавров всех форм обучения направлений 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия» и всех желающих изучить вопросы проектирования программно-информационных систем.

Библиогр.: 7 назв.

Жизненный цикл, модель, анализ, проектирование, реализация, UML, архитектура, поведение, класс, сценарий, автомат, компонент.

Рецензент: кафедра вычислительной и прикладной математики Рязанского государственного радиотехнического университета (зав. кафедрой д-р техн. наук, проф. Г.В. Овечкин)

UML-моделирование функциональности и поведения
программно-информационных систем

Составители Б е л о в Владимир Викторович
Ч и с т я к о в а Валентина Ивановна

Подписано в печать XX.XX.2022. Усл. печ. л. 3.
Тираж 1 экз.
Рязанский государственный радиотехнический университет
390005, Рязань, ул. Гагарина, 59/1.
Редакционно-издательский центр РГРТУ.

СОДЕРЖАНИЕ

ОБЩИЕ ЗАМЕЧАНИЯ

Как готовиться к занятиям

При подготовке к практическому занятию необходимо ознакомиться со списком вопросов по изучаемой теме и самостоятельно на них ответить, используя конспект лекций и рекомендуемую литературу. На практических занятиях студент отвечает на эти вопросы, на вопросы преподавателя, касающиеся теоретического материала, относящегося к данной теме, комментирует полученные в ходе работы результаты. Выполнение задания к практическим занятиям используется для допуска к соответствующим лабораторным работам. Каждая изученная тема должна быть защищена. Предпочтительной формой защиты является публичный доклад с демонстрацией элементов отчёта.

Заметим, практические и лабораторные занятия нацелены на подготовку к выполнению курсового проекта, в рамках которого предстоит оформить принятые проектные решения в виде системы моделей. Существенной частью каждой модели являются диаграммы, но они не являются самостоятельной формой выражения проектного решения.

Запоминаем: проектное решение выражается моделью, а не диаграммой.

Как оформляются отчёты

Все создаваемые диаграммы должны быть тщательно документированы. Документированию подлежит сама диаграмма (секция Documentation заполняется при отсутствии выделений элементов диаграммы) и *все* её элементы – документируются все узлы (геометрические фигуры) диаграммы и все стрелки.

Материалы документирования представляются (в электронном виде) в отчетах по практическим занятиям. Эти материалы желательно подразделять на структурные части с такими названиями как «Документирование акторов», «Документирование прецедентов», «Документирование действий и деятельностей», «Документирование состояний», «Документирование отношений», «Документирование переходов».

Отчёты по изучаемым темам оформляются в виде doc- или docx-документов текстового процессора Word с титульным листом по прилагаемому образцу. В отчёты включаются сведения о проделанной работе: 1) название изучаемой темы; 2) формулировка задания; 3) результаты экспорта построенных диаграмм в графические файлы; 4) тексты документирования диаграмм; 5) описания потоков реализации прецедентов.

Обратим внимание на то, что теоретические сведения и методические указания в отчётах приводить не следует. Теорию необходимо знать, а требования, содержащиеся в методических указаниях, необходимо выполнять.

Какие понятия являются наиболее важными

В рамках изучаемой дисциплины должны быть чётко уяснены понятия, приведённые в таблице 1. При этом следует иметь в виду, что почти все термины являются многозначными и их смысл определяется контекстом. Например, термин «Процесс» в 90 % применений символизирует проблемное начало, т. е. означает название некоторой задачи. Примеры: «Процесс обучения», «Процесс вычисления площади треугольника», «Процесс планирования». В других контекстах термин «Процесс» символизирует способ решения проблемы, т. е. означает последовательность действий во время решения задачи. В этом случае он синонимичен терминам «Алгоритм», «Сценарий», «Деятельность».

Следует понимать, что существуют логический (идейный, семантический, смысловой) и физический (материальный) уровни представления понятий. Например, термин «Программа» в зависимости от контекста рассмотрения означает: 1) алгоритм, представляемый программой; 2) код программы; 3) файл с кодом программы. При этом первые два указанных смысла термина «Программа» соответствуют логическому уровню, т. е. уровню сознания разработчика, а последний смысл – физическому, т. е. уровню физического носителя информации.

Таблица 1 – Первичные понятия моделирования функциональности и поведения программно-информационной системы

Аспект рассмотрения процесса моделирования	**Проблемное начало**	**Способ решения проблемы**	**Форма представления способа решения, делания**
Суть аспекта рассмотрения	Указание того, ***что*** решается, делается	Указание того, ***как*** решатся, делается	Воплощение идей в материально передаваемые формы
Основные термины аспекта рассмотрения	Задача = = Работа = = Процесс	Алгоритм = = Сценарии = = Деятельность	Схема алгоритма, диаграмма деятельности
Вид модели	Функциональная модель	Модель поведения (идейный уровень)	Модель поведения (уровень физического представления)
Выражение в UML	Диаграмма вариантов использования (Use Case Diagram)	Семантика диаграмм поведения	Диаграммы поведения: ■ деятельности; ■ автомата; ■последовательности; ■коммуникации

Выбор предметной области

Для наиболее продуктивного выполнения лабораторных работ рекомендуется выбрать собственную предметную область – объект автоматизации. Желательно использовать «живую» предметную область, – прежде всего, **тему собственного квалификационного исследования** или информационно-материальную среду реального бизнеса – производственного, торгового, сервисного, строительного и т. д. Рекомендуется обратиться за помощью к родственникам, близким и знакомым, имеющим отношение к организации и/или управлению работой предприятий любого уровня. Помощь могут оказать и рядовые, но не равнодушные исполнители бизнес-процессов на любом предприятии.

Предметную область можно также заимствовать и из ранее выполненного курсового проекта, например, по базам данных.

И наконец, темы заданий к лабораторным и практическим работам можно выбрать из списка в Приложении Б.

Перечень выполняемых работ

0. Инструментальная среда моделирования программно-информационных систем StarUML.
1. Диаграмма вариантов использования.
2. Диаграмма деятельности.
3. Диаграмма автомата.
4. Диаграмма классов.
5. Диаграмма последовательности.
6. Диаграмма коммуникации.
7. Диаграмма компонентов.
8. Диаграмма размещения (развертывания) – учебники, в первую очередь [1, 3].

Организационные указания

Для выполнения практических заданий реализуем следующие действия.

1. Устанавливаем пакет StarUML, используя ссылку [8].
2. Работы №№ 1 – 6 выполняем, используя настоящие указания и учебное пособие [4].
3. Работы №№ 7 – 8 выполняем для выбранной предметной области, используя учебники из библиографического списка, в первую очередь [1, 3].

Дополнительное указание

Учебное пособие [4] содержит многочисленные ошибки и методические огрехи. В частности, оно содержит две главы с номером 6. Поэтому к нему следует относиться критически и воспринимать её, главным образом, как руководство по интерфейсу пакета StarUML.

1 ИЗУЧАЕМАЯ ТЕМА «ДИАГРАММА ПРЕЦЕДЕНТОВ»

1.1 Ассоциированные понятия

Вариант использования = Use Case = Прецедент – это некоторая Функция = Задача = Работа = Процесс, реализующий Функцию (Задачу, Работу)

На диаграмме прецедент – это овал с названием Функции (Задачи, Работы).

Осознаём и запоминаем:
формулировка Функции (Задачи, Работы) одновременно является и **названием** Процесса, реализующего Функцию (Задачу, Работу)

Например, «Решение алгебраического уравнения» – это одновременно и формулировка задачи, и название процесса решения уравнения.

По этой причине диаграммы вариантов использования можно называть и функциональными, и процессными (поведенческими). Дополнительным основанием называть диаграммы вариантов использования поведенческими является то, что для прецедентов создаются потоки (*текстовые алгоритмы*) реализации, размещаемые в текстовых файлах, прикрепляемых к прецедентам редактором вложений в окне закладки с именем Attachment.

Варианты использования (Use Case) в отечественной практике моделирования часто для краткости называют прецедентами (в документации по UML слова precedent нет, но оно является синонимом слова case – случай, вариант). Представляется рациональным следующее правило использования терминов «вариант использования» и «прецедент»:

а) все работы (процессы), представляемые на диаграмме вариантов использования в виде овалов, могут называться прецедентами;

б) прецеденты, ассоциированные с акторами-потребителями функциональности, предпочтительнее называть «вариантами использования», поскольку они реально являются именно вариантами использования системы акторами;

в) прецеденты, дополняющие функциональность вариантов использования и/или других (базисных) прецедентов, предпочтительнее называть – просто «прецедентами» или «вспомогательными прецедентами».

Актор (актёр) – сторонняя, т. е. не входящая в моделируемую систему сущность, потребляющая функциональность системы, либо предоставляющая собственную функциональность в порядке дополнения функциональности прецедентов моделируемой системы.

Отношения – это формальные выражения различных связей между акторами, прецедентами, акторами и прецедентами.

Возможные виды отношений на диаграммах вариантов использования:

1) между акторами – только генерализация (обобщение);
2) между прецедентами:
   а) зависимости (include, extend);
   б) генерализация;
3) между акторами и прецедентами – только ассоциации:
   а) направленные ассоциации;
   б) неправленые ассоциации.

Запоминаем: между прецедентами отношений ассоциации нет!

Объединять отношения include и extend в понятие отношений зависимости принято только *в практике* UML-моделирования, в документации же по UML об отношениях include и extend говорится как о самостоятельных видах отношений. По сути дела, отношения include и extend являются конкретными воплощениями отношения «часть – целое» во множестве функциональностей прецедентов. При этом «целым» является функциональность базисного прецедента, а «частями» – функциональности прецедентов, дополняющих эту функциональность – включаемых (include) или расширяющих (extend) прецедентов.

Главное концептуальное различие include- и extend-прецедентов состоит в том, что функциональность include-прецедента используется *безусловно* в процессе реализации базисного прецедента, а функциональность extend-прецедента используется в процессе реализации базисного прецедента только при выполнении некоторого *условия*.

В силу обязательности (неизбежности) применения Include-прецедент является подлинной (неотъемлемой) частью базисного прецедента, – без него базисный прецедент не является полноценным (is not meaningful). В то же время, в силу применения в частном случае (только при выполнении некоторого условия) отдельный extend-прецедент является необязательной добавкой к базисному прецеденту. Одновременно с этим очень часто отсутствие *всех* необязательных добавок может оказаться недопустимым – привести к потере смысла базисного прецедента. Например, базисный вариант использования «Решить уравнение» может расширяться прецедентами «Аналитическое решение» и «Числовое решение». Любой из прецедентов, представляющих методы решения, может отсутствовать – смысл базисного прецедента при этом не потеряется, но отсутствие обоих вспомогательных прецедентов полностью лишает смысла базисный вариант использования.

Пример отношений зависимости приведён на рисунке 1.1. Аналогичный пример, но уже с конкретной семантикой прецедентов показан на рисунке 1.2.

Обратим внимание на чередование применения причастий по отношению к базисному прецеденту:

extending (расширяющий – вспомогательный прецедент)
extended (расширяемый – базисный прецедент)

vs

included (включаемый – вспомогательный прецедент)
including (включающий – базисный прецедент)

Обратим внимание также на то, что словосочетание «base Use Case» рациональнее переводить как «базисный Вариант Использования» в силу следующей тонкости семантик слов базовый и базисный в русском языке: базовый – служащий основой, фундаментом чего-либо; базисный – главный, центральный, важный, используемый для сравнения.

Выше размещён призыв запомнить, что прецедент – это некоторая функция, реализуемая некоторым процессом (программой). Поэтому следует осознать и запомнить: название прецедента может быть только названием процесса. Оно не может являться названием видов информации, таких как Отчёт, Паспортные данные, Результаты поиска. Оно не может также являться и названием событий или условий, таких как Заказ оплачен, Данные введены, Произошла ошибка.

Рисунок 1.1 – Отношения включения и расширения
на диаграммах вариантов использования

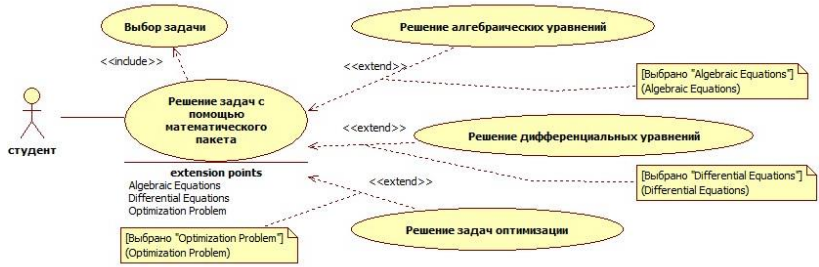


Рисунок 1.2 – Пример диаграммы Use Case с конкретной семантикой прецедентов

1.2 Назначение диаграмм вариантов использования

Диаграммы вариантов использования используются для моделирования функциональности программно-информационной системы, – названия прецедентов представляют собой названия функций (задач, работ), реализуемых самой системой или с её помощью. В то же время, во многих источниках диаграммы вариантов использования категорируются как диаграммы поведения, т. е. как описания процессов, – *последовательностей* действий. Следует понимать, что основанием для этого является только следующее: 1) названия прецедентов – это не только названия функций, но одновременно и *названия* процессов (например, «Решение уравнения» – это одновременно и название задачи, и название процесса); 2) графические составляющие диаграммы вариантов использования дополняются описаниями потоков поведения (потоков состояний), представляющих собой процессы – алгоритмы и сценарии реализации прецедентов.

Описания потоков поведения в обязательном порядке должны создаваться
для *каждого базисного* варианта использования,
представленного на *основной* диаграмме прецедентов.

Алгоритмы вспомогательных прецедентов оформляются в виде подпотоков. Тексты этих подпотоков могут не приводятся в тех случаях, когда их реализация очевидна и/или исчерпан лимит на объём отчёта по теме (30 страниц).

1.3 Руководящие принципы построения диаграммы прецедентов

Диаграмма вариантов использования – это, по сути дела, не одна, а *несколько* диаграмм: одна из них – основная, а остальные – декомпо-

зиционные, – структурирующие варианты использования, представленные на основной диаграмме. Создавая диаграмму прецедентов, следует иметь в виду, что каждый прецедент диаграммы в последующем будет иметь программную реализацию, поэтому детализация прецедентов, т. е. степень подробности описания того, что подлежит программному воплощению, должна быть достаточной для этого программного воплощения.

Основная диаграмма содержит относительно независимые функции моделируемой системы – варианты её использования. Таких вариантов в учебных проектах, как правило, бывает немного – от трёх до девяти. Например, «Регистрация клиента», «Вход клиента в систему», «Поиск и выбор товара», «Оформление продажи», «Учет клиентов, товаров и продаж». На основной диаграмме не принято осуществлять даже частичную декомпозицию вариантов использования. При правильном определении вариантов использования ИС на основной диаграмме, как правило, отсутствуют отношения зависимости между вариантами, что свидетельствует о независимости этих вариантов. В приведённом ниже примере описания функциональности системы «Интернет-магазин» все варианты использования *функционально* (алгоритмически) независимы.Они связаны только *информационно* – «Учет клиентов, товаров и продаж» использует *данные*, порождённые «Регистрацией клиента» и «Оформление продажи». Данные на диаграммах вариантов использования не отображаются основными изобразительными средствами, но крайне полезно показать информационную взаимосвязь вариантов с помощью средств пояснения –аннотациями типа «Text», «Note» (предпочтительно) и «NoteLink» (для указания к чему относится Note – замечание). Пример информационного дополнения диаграммы вариантов средствами аннотирования приведён на рисунке 1.3.

Следует знать, что для того, чтобы сосредоточить внимание читателя на основных элементах функциональности моделируемой программно-информационной системы на диаграммах часто не указывают вспомогательные прецеденты, в частности, как правило, оказывается скрытой интерфейсная составляющая функциональности. Например, на рисунке 1.3 актор Клиент ассоциирован с тремя вариантами использования. Этот факт автоматически предполагает реализацию функции выбора варианта, необходимого в конкретный момент времени, но эта функция не отображена на диаграмме, она «предполагается» по умолчанию.

Следует помнить, что для каждого пользователя моделируемой системы должен быть создан сценарий взаимодействия этого пользователя с системой (сценарий использования системы), и желательно в не-

скольких формах – в виде текстового описания, ассоциированного с диаграммой прецедентов, а в последующем – в виде диаграммы деятельности или диаграммы автомата, или диаграммы последовательности, или диаграммы коммуникации.

Рисунок 1.3 – Пример информационного дополнения диаграммы вариантов использования средствами аннотирования

Декомпозиционные диаграммы *должны* содержать декомпозируемый прецедент, – образующий логический центр декомпозиционной диаграммы, связанный с прочими её прецедентами отношением зависимости. Пример декомпозиции варианта использования показан на рисунке 1.4.

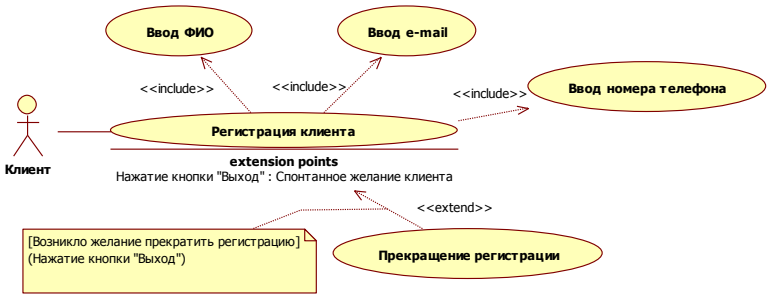


Рисунок 1.4 – Декомпозиция варианта использования «Регистрация клиента»

В методичке Каюмовой [1] на рисунке 21 это правило не выполнено – основной прецедент «Заказ товаров» на диаграмме отсутствует, в результате чего вспомогательные прецеденты превращены в основные, поскольку оказались ассоциированными с актором «Покупатель».

Следует иметь в виду, что очень часто *частью* программно-информационной системы является человек. Обычно, это бывает в следующих ситуациях:

1) на человека возлагается обязанность выполнения некоторой функции на основе интуиции или плохо формализуемого опыта практической деятельности; чаще всего это различного рода консультации пользователей системы, хотя могут иметь место и применения интуиции специалистов в задачах прогнозирования, оценки ситуаций, диагностики и принятия управленческих решений – в тех случаях, когда искусственный интеллект оказывается не достаточно эффективным;

2) на человека возлагается обязанность выполнения рутинных операций, связанных с техническим обслуживанием системы, и ведением баз данных; обычно такого человека называют админом.

Человек-элемент системы неотделим от этой системы, он выполняет часть её функциональности, а не является сторонней сущностью – субъектом, объектом или другой системой, потребляющей или помогающей выполнить функциональность проектируемой системы. Такие сторонние сущности называются акторами. Сторонние системы-помощники отличаются от проектируемой системы происхождением (целью и/или разработчиком и/или местом и/или временем создания), владельцем и механизмом обслуживания. Человек-элемент системы – это не актор, он работник (worker), действующий внутри системы.

Для того, чтобы поместить worker на диаграмму вариантов использования необходимо выполнить следующие действия:

1) навести курсор на элемент Use Case View;
2) нажать правую клавишу мыши;
3) в появившемся контекстном меню выбрать Add;
4) в новом появившемся контекстном меню выбрать Class; созданный класс отобразится в навигаторе модели Model Explorer на уровне диаграмм с именем по умолчанию;
5) в окне свойств Properties изменить имя созданного класса на требуемое, например, Админ, Аналитик, Продавец;
6) перетащить методом Drag and drop созданный класс из навигатора модели на рабочее поле диаграммы;
7) выделить прямоугольник класса щелчком мыши;
8) в окне свойств Properties назначить стереотип worker (последний в списке стереотипов);
9) нажать на выделенном прямоугольнике класса правую клавишу мыши, далее по цепочке из трёх контекстных меню выбрать: Format –> Stereotype Display –> Iconic; прямоугольник класса изменится на человечка в круге;

10) далее через контекстное меню Format можно подавить отображение линий атрибутов и операций worker'а по схеме: Format –> Suppress Attributes и Format –> Suppress Operations.

1.4 Состав выполняемых работ по изучаемой теме

В процессе выполнения задания необходимо выполнить следующие работы.

1. Построить диаграмму вариантов использования первого уровня – основную диаграмму. Рекомендуемая подрисуночная подпись: «Варианты использования проектируемой системы»
2. Построить диаграммы декомпозиции вариантов использования, представленных на диаграмме первого уровня. Рекомендуемые подрисуночные подписи:

«Декомпозиция варианта использования <НАЗВАНИЕ ПРЕЦЕДЕНТА>»

Замечание: конструкция <НАЗВАНИЕ ПРЕЦЕДЕНТА> должна быть заменена на фактическое название декомпозируемого прецедента.

3. Осуществить документирование всех диаграмм и всех их элементов (см. требования к оформлению отчёта) в секции Documentation диаграммы.
4. Составить текстовые описания алгоритмов реализации (потоков состояний) *всех* вариантов использования, представленных на диаграмме первого уровня (основной диаграмме) и прикрепить получаемые текстовые документы редактором вложений в окне закладки с именем Attachment к соответствующим прецедентам.

Замечание: выполнение этого пункта работ косвенно потребует составления текстовых описаний алгоритмов реализации (потоков состояний) всех прецедентов всех (декомпозиционных) диаграмм;

5. Поместить все документирования и описания алгоритмов реализации в отчёт.

Замечание: если для документирования всех элементов всех диаграмм и для описания алгоритмов реализации всех прецедентов требуется более **тридцати** страниц отчёта, то можно ограничиться несколькими диаграммами, могут быть исключены документирования и описания потоков, связанные с некоторыми диаграммами декомпозиции.

1.5 Требования к оформлению основных элементов отчёта

1. *Все* элементы диаграмм – и узлы, и стрелки должны иметь пояснения в секции Documentation диаграммы и в тексте отчёта.
2. Пояснение стрелки должно представлять собой *обоснования* выбранного отношения, представляемого данной стрелкой.

3. Текст пояснений в отчёте желательно подразделять на структурные части – «Документирование акторов», «Документирование прецедентов», «Документирование обобщений | ассоциаций | включений | расширений»

Примерные схемы обоснований:

Для ассоциации между актором и прецедентом:

а) «Отношение ассоциации выбрано направленным от актора <название актора> к прецеденту <название прецедента>, потому что актор является пассивным участником взаимодействия – запускает выполнение прецедента, задаёт исходные данные и получает результат».

б) «Отношение ассоциации между актором <название актора> и прецедентом <название прецедента> выбрано ненаправленным, потому что актор является активным участником взаимодействия – не только запускает выполнение прецедента, задаёт исходные данные и получает результат, но и участвует в диалоге с программой, реализующей прецедент».

Для зависимости между прецедентами:

а) «Тип отношение зависимости между прецедентами <название 1-го прецедента> и <название 2-го прецедента> выбран «include», потому что при реализации прецедента <название 1-го прецедента> функциональность прецедента <название 2-го прецедента> используется безусловно.

б) «Тип отношение зависимости между прецедентами <название 1-го прецедента> и <название 2-го прецедента> выбран «extend», потому что при реализации прецедента <название 1-го прецедента> функциональность прецедента <название 2-го прецедента> используется только при выполнении условия <формулировка условия>.

Замечание: условие <формулировка условия> обязательно помещается в свойство (property) Condition поясняемой стрелки отношения «extend».

4. *Все* варианты использования, представленные на диаграмме первого уровня (основной диаграмме), должны иметь текстовый алгоритм реализации (спецификацию потоков поведения) в текстовом документе, прикрепляемом редактором вложений в окне закладки с именем Attachment.
5. Текстовые описания алгоритмов реализации прецедентов оформляются в виде таблиц. Примеры различных вариантов оформления текстовых алгоритмов приведены в [7].

В настоящем пособии в качестве примеров текстовых описаний алгоритмов реализации прецедентов приведены: таблица 2 – сценарий использования системы Интернет-магазин клиентом и таблица 3 – сце-

нарий вычисления первого замечательного предела.

Основной поток алгоритма представляет собой последовательность действий и ссылок на деятельности. Действия и ссылки нумеруются натуральными цифрами в порядке их выполнения. При этом ссылки имеют специфический синтаксис: за номером в основном потоке размещается номер ссылки с префиксом, указывающим на тип ссылки – E, A или S. Идентификаторы, начинающиеся на букву A, символизируют альтернативные потоки, т. е. потоки, выполняемые при справедливости некоторых условий. Идентификаторы, начинающиеся на букву S, символизируют подпотоки.

Различия между альтернативными потоками и подпотоками заключаются в следующем. Альтернативные потоки – это альтернативные ветви алгоритма реализации одного прецедента. Они всегда обусловлены. Ссылка на альтернативный поток имеет синтаксис: **k. Am [c],** где **k** – номер ссылки на альтернативный поток в основном потоке; **m** – номер альтернативного потока; **c** – условие, при котором выполняется альтернативный поток. Пример ссылки на альтернативный поток:

5. A2. [Выбран Выход из системы]

Альтернативные потоки располагаются в одной таблице с основным потоком, но в отдельной секции, – чтобы не нарушать читаемость основного потока.

Подпотоки – это предопределённые алгоритмы по терминологии схем алгоритмов. Их тексты располагаются в отдельных таблицах. Они используются для описания реализаций вспомогательных прецедентов. Ссылки на них могут обусловливаться, но могут и не иметь условий обращения к предопределённому алгоритму. Обусловленные ссылки соответствуют extend-зависимостям, а безусловные – include-зависимостям между прецедентами. Ссылка на подпоток реализации extend-зависимого прецедента имеет синтаксис: **k. Sm [c]**. Ссылка на подпоток реализации include-зависимого прецедента **k. Sm** аналогична, но не содержит условия.

Идентификаторы, начинающиеся на букву E, символизируют потоки ошибок, которые также выполняются при справедливости некоторых условий, но эти условия не соответствуют рабочему развитию процесса. Это обработчики исключительных ситуаций. Текст потока ошибок при малом его объёме может размещаться в одной таблице с основным потоком, но может оформляться и в виде отдельной таблицы. Ссылка на поток ошибок имеет синтаксис **k. Em [c]**

Заметим, что конструкции **Am [c], Sm [c], Sm, Em [c]** без номера ссылки в потоке реализации используются как заголовки описаний альтернативных, подпотоков и потоков ошибок в разделах их описаний.

Таблица 2 – Сценарий использования системы «Интернет-магазин»

Название:	Сценарий использования системы «Интернет-магазин»
Действующие лица:	Клиент
Краткое описание:	Клиент выбирает и реализует необходимую функцию Системы. В процессе общения с Системой каждое интерфейсное окно предоставляет возможность досрочного завершения общения с Системой.
Предусловия:	У клиента возникла потребность воспользоваться системой удалённого приобретения товаров. Он запустил Систему.
Постусловия:	Клиент завершил общение с Системой
Основной поток (нормальное течение):	1. Система отображает окно с возможными вариантами действий: ✓ Регистрация клиента // выполняется при первом входе в систему ✓ Вход в систему // выполняется при последующих входах в систему ✓ Выход из системы // потенциальная возможность поведения клиента 2. Пользователь выбирает необходимое действие 3. S1. [Выбрана Регистрация клиента] // подпоток регистрации 4. A1. [Выбран Вход в систему] 5. A2. [Выбран Выход из системы] 6. Выполнение прецедента завершается
Альтернативные потоки (альтернативные течения):	A1. [Выбран Вход в систему] 1. Система отображает окно с предложением ввести краткую авторизационную информацию (либо пароль, либо e-mail, либо номер телефона) 2. A2. [Выбран Выход из системы] 3. Система отображает окно с предложением приступить к поиску и выбору товаров и возможностью выхода из Системы 4. A2. [Выбран Выход из Системы] 5. S2. [Выбран Поиск и выбор товаров] A2. [Выбран Выход из системы] 1. Переход на пункт 6 основного потока
Потоки ошибок:	Потоки не определены
Подпотоки	Подпотоки S1 и S2 описываются отдельными таблицами
Приоритет (Критично | Важно | Желательно):	Критично
Частота использования (Всегда | Часто | Иногда | Редко | Один раз):	Всегда

Таблица 3 – Описание прецедента «Первый замечательный предел»

Название:	Решение уравнения
Действующие лица:	Пользователь
Краткое описание:	Вычисляет формулу первого замечательного предела $y = \lim_{x \to 0} \left(\frac{\sin(x)}{x} \right)$
Предусловия:	У пользователя возникла потребность вычислить формулу первого замечательного предела. Он запустил Систему.
Постусловия:	Система отображает решение
Основной поток (нормальное течение):	1. Система отображает предложение ввода аргумента x 2. Пользователь вводит значение x 3. E1. [Ошибка ввода] 4. A1. [$x = 0$] 5. A2. [$x \neq 0$] 6. Вывод x, y 7. Выполнение прецедента завершается
Альтернативные потоки (альтернативные течения):	A1. [$x = 0$] 1. $y = 1$ 2. Переход на пункт 6 основного потока A2. [$x \neq 0$] 1. $y = \frac{\sin(x)}{x}$ 2. Переход на пункт 6 основного потока
Потоки ошибок:	E1. [Ошибка ввода] 1. Переход на пункт 1 основного потока
Приоритет (Критично | Важно | Желательно):	Критично
Частота использования (Всегда | Часто | Иногда | Редко | Один раз):	Всегда

2 ИЗУЧАЕМАЯ ТЕМА «ДИАГРАММА ДЕЯТЕЛЬНОСТИ»

2.1 Ассоциированные понятия

Поведение (Behavior) ИС – совокупность всех процессов, реализуемых в ИС.

Процесс (дискретный) – последовательность операций.

Целенаправленные процессы – процессы, направленные на достижение некоторой цели (получение некоторого результата).

Форма представления целенаправленных процессов – алгоритмы и сценарии.

Графические средства представления алгоритмов и сценариев – диаграммы.

Основные диаграммы, используемые для моделирования поведения (процессов = работ):

D1. State machine Diagram = State chart Diagram = Диаграмма автомата
D2. Activity Diagram = Диаграмма деятельности.
D3. Sequence Diagram = Диаграмма последовательности
D4. Communication Diagram = Collaboration Diagram = Диаграмма коммуникации / кооперации (сотрудничества)

D1 $\subseteq$ D2 – диаграммы автомата и деятельности имеют общую теоретическую платформу – Сети Петри; по совокупности изобразительных средств диаграммы деятельности являются подмножеством диаграмм автомата.

D3 = D4 – диаграммы последовательности и коммуникации семантически эквивалентны.

Процессы представляют собой цепочки (последовательности) действий и деятельностей. По определению:

Действие = Action – семантически простая (элементарная, неделимая, не имеющая структуры) краткосрочная *непрерываемая* операция.

Деятельность = Activity – *прерываемая* последовательность действий. Прерывание деятельности возможно при переходе от одного действия к другому.

В StarUML и действия, и деятельности с известными читателю алгоритмами реализации в диаграммах изображаются с помощью одного и того же инструмента «ActionState». Деятельности же с неизвестными алгоритмами реализации *в местах их использования* представляются *ссылками* на деятельности. Ссылки создаются с помощью инструмента «SubactivityState». Для каждой такой ссылки необходимо создать отдельную (дополнительную) диаграмму реализации.

Ссылку на деятельность можно называть обращением к деятельности или вызовом деятельности – всё как в программировании: в программе (или подпрограмме более высокого уровня) размещается обра-

щение к подпрограмме, называемое также вызовом подпрограммы или ссылкой на подпрограмму, а в отдельном месте программного кода размещается текст реализации (определение) подпрограммы.

ActionState и SubactivityState в диаграммах изображаются в виде овалов с названиями представляемых работ и деятельностей, см. рисунок 2.1. Овал SubactivityState имеет специальную пиктограмму (сосиску или трезубец) в правом нижнем углу.

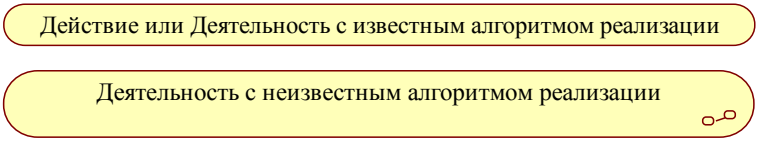


Рисунок 2.1 – Изображение действий, и ссылок на деятельности

Запоминаем факт:

деятельность может быть конечной или бесконечной во времени.

Бесконечная деятельность – это ожидание некоторого события, называемого триггером (триггерным событием). Во время ожидания возможно отображение различного рода инструкций и/или рекламы.

Диаграммы деятельности следует помещать в представление Logical View.

В контексте применения инструмента SubactivityState в StarUML следует знать следующее. Когда модельер создаёт на уровне представления (Logical View) новую диаграмму деятельности, то фактически в модели создаётся деятельность, представляющая собой механизм объединения диаграммы деятельности и её элементов. Условно можно изобразить так.

Деятельность = Диаграмма деятельности + Элементы диаграммы

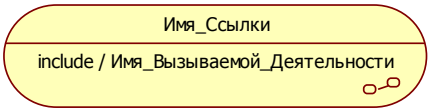
Понять эту формулу легче, если осознать, что диаграмма деятельности, по сути дела, – это механизм рисования (интегратор) графических элементов из некоторой совокупности этих элементов. Сама диаграмма – это выразитель, средство представления деятельности.

В StarUML версии 5.0.2.1570 эта деятельность по умолчанию имеет имя ActivityGraph*n*, где *n* – целое число. В версиях StarUML 3.x и 4.x она называется просто Activity. На уровне ActivityGraph позиционируются диаграммы деятельности, используемые для представления алгоритмов реализации варианта использования (некоторой задачи). В StarUML 5.0.2.1570 внутри интегратора ActivityGraph создаётся ещё один интегратор – с именем TOP, объединяющий элементы всех диаграмм, объединяемых интегратором ActivityGraph. В версиях StarUML

3.x и 4.x интегратор TOP не используется, и элементы диаграмм позиционируются вместе с диаграммами на уровне Activity.

Именование ссылок и детальностей

В отличие от многих языков программирования в StarUML ссылка на что-либо (например, на деятельность или автомат) имеет *собственное* имя, а то, на что ссылаются, тоже имеет собственное имя. В StarUML версии 5.0.2.1570 ссылка выглядит так:



В версиях StarUML 3.x и 4.x и в стандарте языка ссылки на деятельность имеют иной вид:

Имя_Ссылки:Имя_Вызываемой_Деятельности

Применение имён ссылок позволяет различать ссылки на одну и ту же деятельность (на одну и ту же функциональность) в рамках одного алгоритма или нескольких алгоритмов. Например, две следующие ссылки

Выч_Площади_Под_Домом:Вычисление_Пощади_Многоугольника

Выч_Площади_Под_Садом:Вычисление_Пощади_Многоугольника

вызывают одну и ту же деятельность «Вычисление_Площади_Многоугольника».

Если потребности в различении разных ссылок на одну и ту же деятельность нет, то «Имя_ссылки» и связанное с ним «Имя_Вызываемой_Деятельности» часто представляют схожими именами, например, «регКлиента» и «РегистрацияКлиента», – первое слово в имени ссылки сокращается и начинается со строчной буквы.

Именование деятельностей и их диаграмм

Имя диаграммы может совпадать с именем деятельности, представляемой этой диаграммой или отличаться от него несущественно, например, префиксом D_, как в следующем примере: «D_ Вычисление_Пощади_Многоугольника», но языковые проблемы инструментов UML-моделирования обусловливают следующую особенность: имена диаграмм в деятельностях целесообразно записывать на английском языке по аналогии с «D_ Calculate_Polygon_Area».

2.2 Назначение диаграмм поведения

Диаграммы поведения используются для моделирования процессов в программно-информационной системе, и представляют собой алгоритмические реализации прецедентов, представляемых на диаграмме вариантов использования.

Диаграммы поведения в обязательном порядке должны создаваться для *каждого базисного* варианта использования, представленного на *основной* диаграмме прецедентов.

Алгоритмы вспомогательных прецедентов приводятся в тех случаях, когда их реализация не очевидна и/или не осуществлялась ранее в компании разработчика.

Заметим, что термин «алгоритмическая реализация» применяется редко, чаще говорят просто «алгоритм», ещё чаще – «сценарий». Термин «сценарий» в эпоху UML 1 означал конкретную траекторию в алгоритме – прохождение алгоритма по конкретным ветвям, представляемое диаграммой последовательности. Поэтому, когда говорили «сценарий», подразумевали диаграмму последовательности, а когда говорили «диаграмма последовательности» подразумевали сценарий.

В настоящее время комбинированные фрагменты (Combined Fragment) превращают диаграммы последовательности в практически полноценное средство представления алгоритмов, поэтому термин «сценарий» стал практически эквивалентным термину «алгоритм».

2.3 Руководящие принципы построения диаграммы деятельности

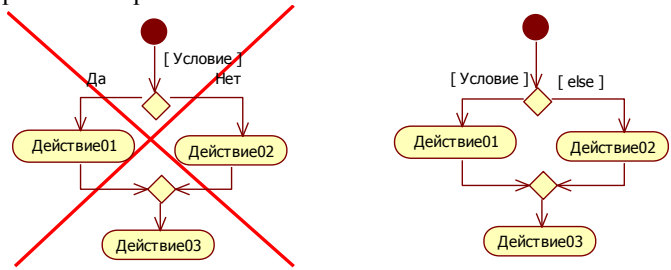
Диаграммы деятельности создаются для *каждого* варианта использования, представленного на *основной* диаграмме прецедентов. Как и диаграмма вариантов использования, диаграмма деятельности – это не одна, а *несколько* диаграмм: одна из них – основная, а остальные – декомпозиционные, – структурирующие *деятельности*, представленные на основной диаграмме и других декомпозиционных диаграммах.

2.3.1 Оформление ветвлений

При построении диаграмм деятельности следует избегать ошибки, содержащейся в некоторых методических указаниях, например, в [4] и в интернет-источниках – использования слов «Да» и «Нет» над стрелками, исходящими из узла решения. Эта практика является вольным переносом нотации блок-схем алгоритмов в диаграммы UML, что совершенно недопустимо. Дело в том, что в UML узел решения – это будущий оператор `switch` в С-подобных или `case` в Паскале-подобных языках (в узел решения может входить и из него может исходить любое количество стрелок). Над исходящими стрелками следует размещать условия, при этом над одной из стрелок рекомендуется размещать английское слово else. По этой стрелке будет осуществлён переход в том случае, если ни одно из условий над другими стрелками не будет выполнено. Образно можно считать, что стрелка исчезает из диаграммы во

время исполнения алгоритма, если ассоциированное с этой стрелкой условие не выполняется. Если же ассоциированное со стрелкой условие выполняется, то стрелка сохраняется и передаёт поток управления.

Следует знать, что условия и else набираются в свойствах (Properties) стрелок, а именно в свойствах «GuardCondition», называемых в некоторых реализациях UML просто «Condition» или «Guard». Набираемые в свойствах условия появляются над стрелками автоматически и в квадратных скобках. Изложенное замечание наглядно иллюстрируется рисунком 2.2. Рисунок 2.2 иллюстрирует также тот факт, что узел решения в UML является не только средством ветвления, но и средством слияния стрелок. В блок-схемах такого средства нет – стрелки соединяются простым соприкосновением.



а) не верное б) верное

Рисунок 2.2 – Изображение ветвления в UML

2.3.2 Оформление разделов активности (Activity Partitions) в нотации плавательных дорожек (Swimlanes)

При использовании в диаграмме разделов активности в виде плавательных дорожек следует иметь в виду следующее.

Для исполнителей деятельностей, являющихся акторами-людьми, следует учитывать, что человек, работая с любой программой, может по сути дела (физически) только нажимать на клавиши, пользоваться мышкой, думать и принимать решения. В то же время, в диаграммах деятельности принято указывать не «физику» манипуляций человека, а логическое содержание действий, выполняемых этим человеком – «Ввод исходных данных», «Ввод критерия поиска», «Переход к разделу каталога», «Исследование товара», «Помещение товара в корзину» и т. п. Однако, при этом, крайне желательно представлять в диаграмме механизм, используемый человеком для реализации необходимых действий, например, интерфейсное окно, отображаемое программной системой, как это показано на рисунке 2.3, представляющем собой пример диаграммы деятельности с разделами активности в виде плавательных дорожек.

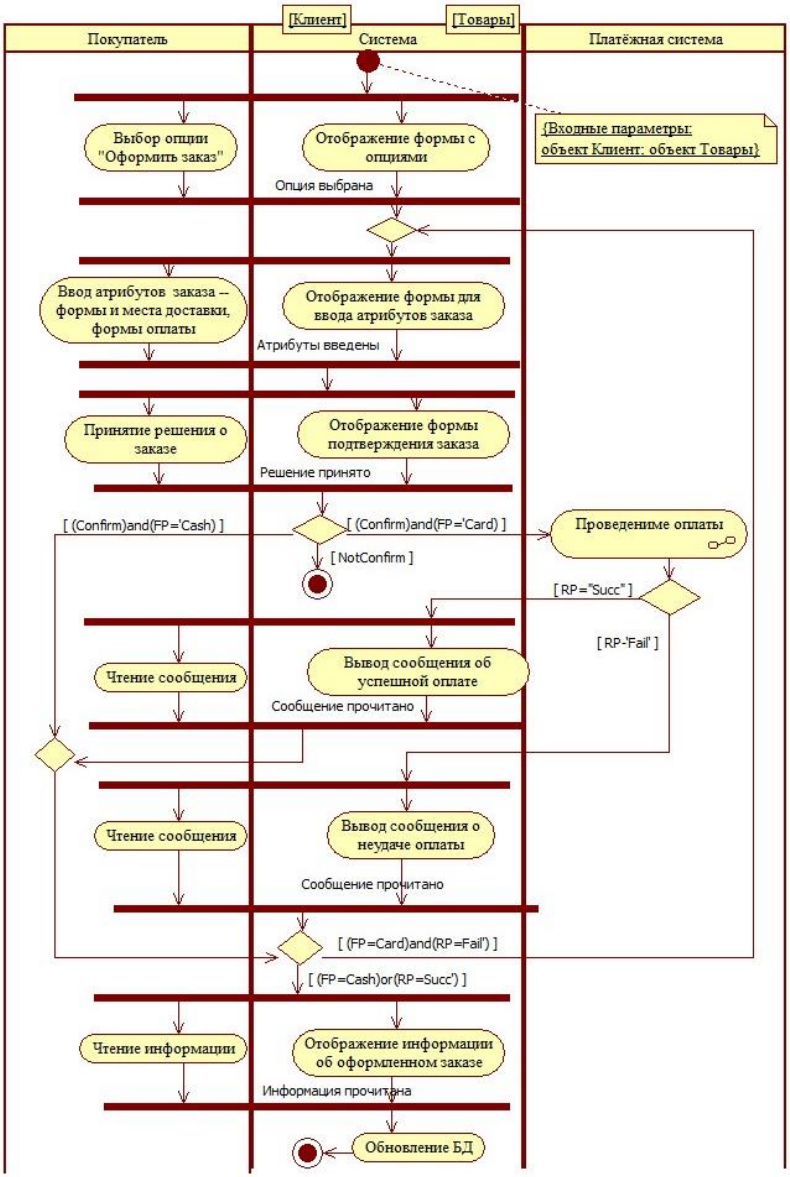


Рисунок 2.3 – Алгоритм реализации деятельности «Оформление заказа»

Следует также иметь в виду, что «Система», показанная на рисунке 2.3, интегрирует в себе и серверную и клиентскую (в виде браузера) части программно-информационной системы. Это позволяет избежать излишней детализации действий клиенткой части приложения, не приводя такие действия как «Получение данных», «Рендеринг окна» и т. п. Такой подход вполне приемлем, если он соответствует текущему контексту моделирования.

2.4 Состав выполняемых работ по изучаемой теме

В процессе выполнения задания необходимо выполнить следующие работы.

1. Построить диаграммы деятельности для каждого варианта использования, представленного на диаграмме первого уровня. Рекомендуемая подрисуночная подпись для диаграмм:

   «Алгоритм реализации варианта использования <НАЗВАНИЕ ПРЕЦЕДЕНТА>»

Замечание: конструкция <НАЗВАНИЕ ПРЕЦЕДЕНТА> должна быть заменена на фактическое название реализуемого прецедента.

2. Построить диаграммы деятельности, реализующих ссылки на деятельности (узлы SubactivityState), содержащиеся в диаграммах реализации вариантов использования.

Замечание: по сути дела, первые два пункта состава выполняемых работ повторяют пункт 4 состава выполняемых работ темы «Диаграмма вариантов использования». Отличие состоит только в том, что вместо тестовых алгоритмов создаются их графические представления.

3. Осуществить документирование всех диаграмм и всех их элементов (см. требования к оформлению отчёта) в секции Documentation диаграммы.
4. Поместить все документирования всех диаграмм и их элементов в отчёт.

2.5 Требования к оформлению отчёта

1. Все элементы диаграмм – и узлы, и стрелки – должны иметь пояснения в секции Documentation.
2. Пояснение узлов должно устранять неполноту (incompleteness [ɪnkəmˈpliːtnɪs]) и неоднозначность (ambiguity [æmbɪˈgjuːɪtɪ]) содержания действий и деятельностей, представленного в названии действия / деятельности. Как правило, одно название не может полностью и однозначно выразить смысл действия / деятельности. Следует стремиться к тому, чтобы пояснения действий и деятельностей, помещаемые в Documenta-

tion, обеспечивали представление поясняемых действий и деятельностей в виде программного кода – это требование модели поведения на этапе окончательного (рабочего) проектирования. Модель требований может содержать более размытые (fuzzy [ˈfʌzɪ]) формулировки действий и деятельностей, но, тем не менее, они должны обеспечивать точное представление «физики» реализации действия / деятельности.

3. Каждая ссылка (SubactivityState) на деятельность (Activity), для полного представления смысла которой недостаточно текстового пояснения, должна иметь собственную диаграмму деятельности с алгоритмом реализации деятельности, на которую осуществлена ссылка.
4. Пояснение стрелки должно представлять собой обоснования выбранного набора надписей над данной стрелкой – свойств Triggers, Guard Condition, Effects.

Примерные схемы обоснований:

Для ссылок на деятельности:

а) деятельность «Ожидание ввода данных» (см. рисунок 2.4) семантически тривиальна, поэтому отдельная диаграмма её реализации не приводится;

б) деятельность «Ввод исходных данных» (см. рисунок 2.4) предполагает реализацию стандартной процедуры «Ввод с клавиатуры» | реализацию стандартной процедуры «Ввод из файла» | реализацию стандартного процесса заполнения текстовых полей окна пользовательского интерфейса и выбора элементов раскрывающихся списков. [Среди элементов, перечисленных через знак «|», следует выбрать что-то одно – это и будет конкретным проектным решением.] Данная деятельность семантически проста, поэтому отдельная диаграмма её реализации не приводится.

Заметим, что применение к деятельности терминов «тривиальна», «проста» должно быть обоснованным – подобным приведённым выше примерам.

Для стрелки перехода:

а) стрелка перехода от действия «Вывод приглашения к вводу ИД» к деятельности «Ожидание ввода данных» не содержит никаких надписей, потому что переход осуществляется по завершению деятельности «Вывод приглашения к вводу ИД» и является безальтернативным, т. е. единственным (см. рисунок 2.4).

б) из деятельности «Ожидание ввода данных» имеется два выхода, соответствующих двум триггерным событиям – «Нажато CTRL+C» и «Нажата информационная клавиша»; соответствующие переходы

осуществляются безусловно и без действий на переходах (см. рисунок 2.4)

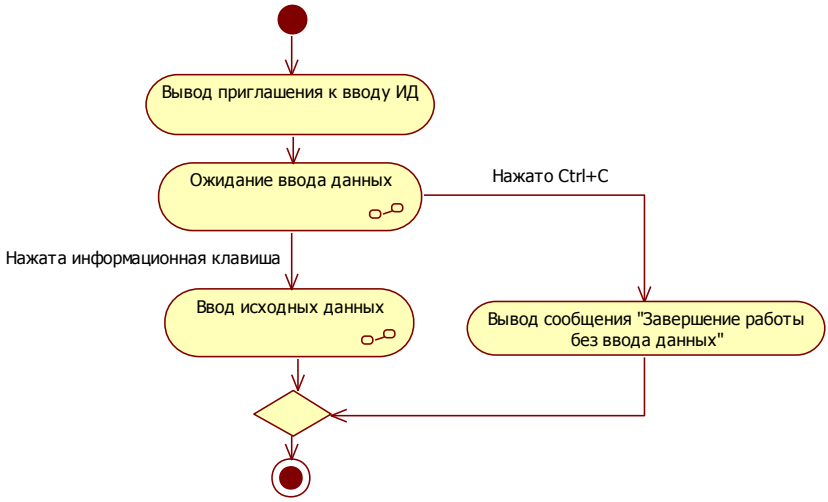


Рисунок 2.4 – Пример выхода из состояния с бесконечной деятельностью по триггерным событиям

В традиционных схемах алгоритмов блок «Процесс» – аналог узла «Действие» (Action) в диаграммах UML – может иметь произвольное количество входов и единственный выход. В диаграммах же деятельности узел «Action» должен иметь один вход и может иметь произвольные количество выходов. При этом следует помнить следующее: один вход в узел «Action» формируется путём слияния стрелок, направленных к данному «Действию» с помощью узла «Решение» (Decision).

Узел «Decision» может иметь произвольное количество входов и выходов (в том числе и единственный), поэтому он одновременно является средством реализации и ветвлений, и слияний.

Выход из узла «Action» может быть осуществлён либо по завершению деятельности, представляемой этим узлом, либо по триггерному событию, возникающему во время выполнения представляемой деятельности. Из узла «Action» может существовать единственный выход только в том случае, если представляемая этим узлом деятельность является завершаемой. Если же деятельность теоретически не завершаема, то выход из узла «Action» возможен только по триггерному событию. Таких событий может быть несколько, и каждому из них соответствует свой выход из выполняемой деятельности. Следует следить за гарантированностью возникновения хотя бы одного из триггерных событий.

3 ИЗУЧАЕМАЯ ТЕМА «ДИАГРАММА АВТОМАТА»

3.1 Ассоциированные понятия

Англоязычное название – State Machine Diagram. Abbreviated form – stm.

Автомат (state machine) – совокупность состояний и переходов между ними.

Состояния: простые (simple), составные (composite), ссылочные (submachine), специальные (pseudo).

Переходы: простые и составные.

Надписи над стрелками: событие перехода (trigger event), сторожевое условие (guard condition), действие на переходе (effect).

3.2 Руководящие принципы построения диаграммы автомата

3.2.1 Общие замечания

Как и любая другая диаграмма, рассматриваемая диаграмма автомата является не самостоятельно значимым элементом проекта, а выражающим некоторое содержательное проектное решение, – определяющее поведение (алгоритм / сценарий функционирования) некоторой части проектируемой системы в процессе реализации конкретного прецедента. Другими словами, диаграмма автомата – это альтернативная форма представления алгоритмической реализации сценариев.

Название реализуемого алгоритма / сценария следует использовать в подрисуночной подписи к диаграмме автомата, как это представлено на последующих рисунках.

Разработку диаграмм автомата следует осуществлять поакторно – вначале реализуется *сценарий использования* проектируемой системы первым актором, за тем со вторым и т.д.

> В рамках реализации сценария реализуются варианты использования, ассоциированные с соответствующим актором.

3.2.2 Принцип иерархии / декомпозиции в диаграммах автомата

Проявление принципа иерархии / декомпозиции в диаграммах автомата полностью аналогична проявлению этого принципа в диаграммах прецедентов и деятельности:

1) вначале изображается укрупненная версия сценария использования системы актором; если на диаграмме вариантов использования актор ассоциирован с несколькими вариантами, то сценарий поведения актора должен включать процесс выбора актором необходимого варианта использования системы; как правило, указанный процесс не отоб-

ражается на диаграмме в виде самостоятельного прецедента, – он подразумевается, носит чисто «технический» характер и примитивен в своей реализации; его отсутствие позволяет сосредоточить внимание читателя на ассоциациях актора именно с вариантами использования системы, не отвлекая внимания на технический процесс выбора варианта; иллюстрирующий пример показан на рисунке 3.1;

2) затем осуществляется алгоритмическая реализация вариантов использования в виде диаграмм автоматов; при этом вспомогательные прецеденты могут представляться и виде деятельностей, как это показано на рисунке 3.2, и в виде состояний, как это показано на рисунках 3.3 и 3.4;

3) далее осуществляется декомпозиция – алгоритмическая реализация вспомогательных прецедентов, если в этом имеется необходимость;

- подлежат алгоритмической реализации и деятельности состояний, чьи названия не исчерпывают семантику выполняемых операций;
- для примитивных деятельностей указывается факт отсутствия их алгоритмических реализаций по причине их семантической простоты;

4) процесс декомпозиции может быть остановлен в случае существенного превышения рационального объёма отчётных материалов.

3.2.3 Примеры диаграмм автомата

Диаграмма, представленная на рисунке 3.1, – пример алгоритмической реализации сценария работы клиента с проектируемой системой.

Сценарий представлен в виде последовательного составного состояния – содержащего в себе один автомат. Особенностью этого автомата является то, что названия его состояний совпадают с названиями деятельностей внутри этих состояний.

Начальное состояние «Отображение интерфейсного окна и ожидание выбора клиента» содержит одноименную внутреннюю деятельность с потенциально бесконечной длительностью, – клиент может медлить с выбором варианта использования системы бесконечно долго, поэтому выход из этого состояния возможен только по триггерным событиям. Таких события три – по числу возможных вариантов использования системы. Сами варианты использования представлены простыми состояниями с одноименными внутренними деятельностями. Автомат отображает возможности клиента чередовать варианты использования системы, переходя через «технические» состояния принятия решений о дальнейших действиях с системой.

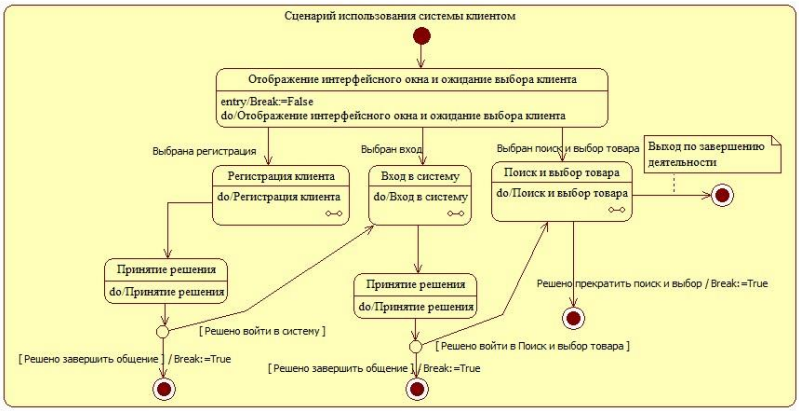


Рисунок 3.1 – Сценарий работы клиента с проектируемой системой

Рисунки 3.2 – 3.4 представляют собой альтернативные решения, которые могут быть приняты дизайнером относительно реализации прецедента «Регистрация клиента». Каждый из указанных трёх вариантов является допустимым, выбор конкретного из них – дело индивидуальных предпочтений проектировщика. Первый вариант реализации (самый естественный) предполагает, что общение клиента с системой во время регистрации осуществляется с помощью одного интерфейсного окна. Состояние «Отображение окна» реализуется компьютером, состояние «Действия клиента», – естественно клиентом.

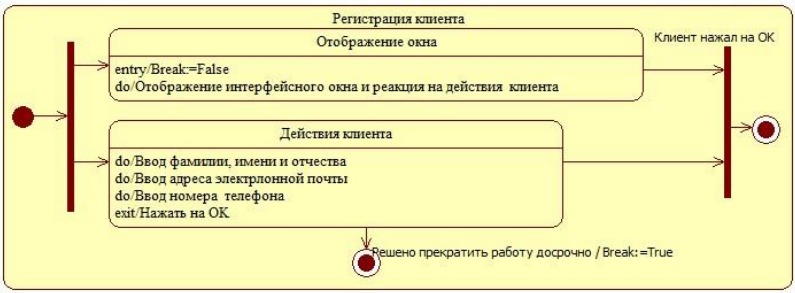


Рисунок 3.2 – Сценарий реализации прецедента «Регистрация клиента» в виде составного состояния. Вариант 1

Второй вариант реализации (несколько искусственный) аналогичен первому, и отличается от него только тем, что требуется строгая очерёдность ввода клиентом сведений о себе.

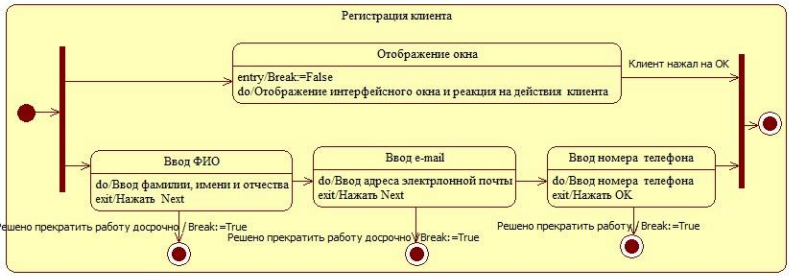


Рисунок 3.3 – Алгоритм реализации прецедента «Регистрация клиента» в виде составного состояния. Вариант 2

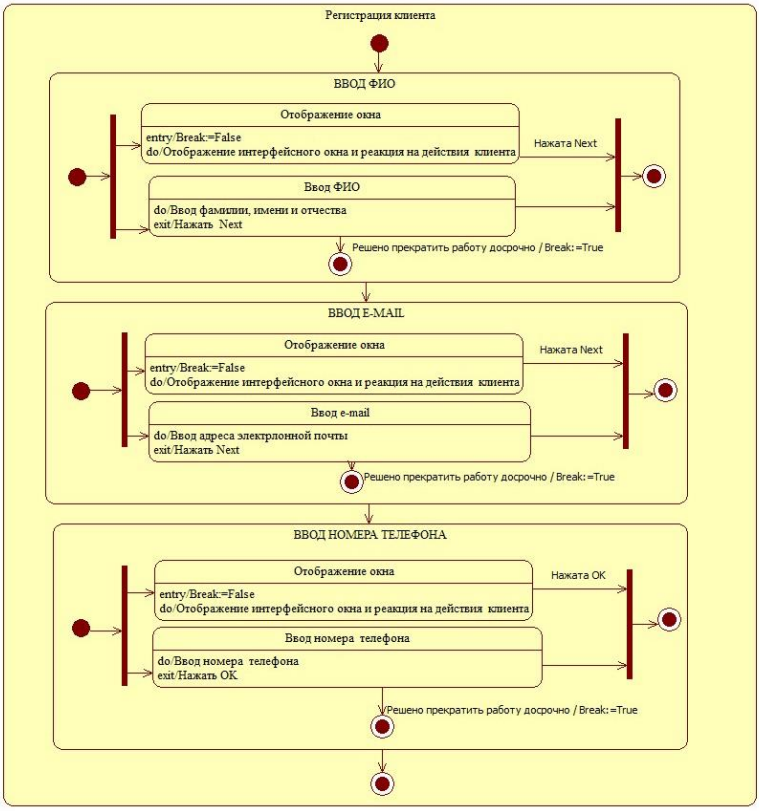


Рисунок 3.4 – Алгоритм реализации прецедента «Регистрация клиента» в виде составного состояния. Вариант 3

Третий вариант реализации аналогичен второму, в нём также требуется строгая очерёдность ввода клиентом сведений о себе, но отличие состоит в том, что ввод каждой группы данных осуществляется с помощью собственных окон.

Алгоритм на рисунке 3.5 иллюстрирует применение внутренних переменных «Break», «Статус», «Оплата». Их следует реализовать как атрибуты вспомогательного объекта в текущем потоке управления.

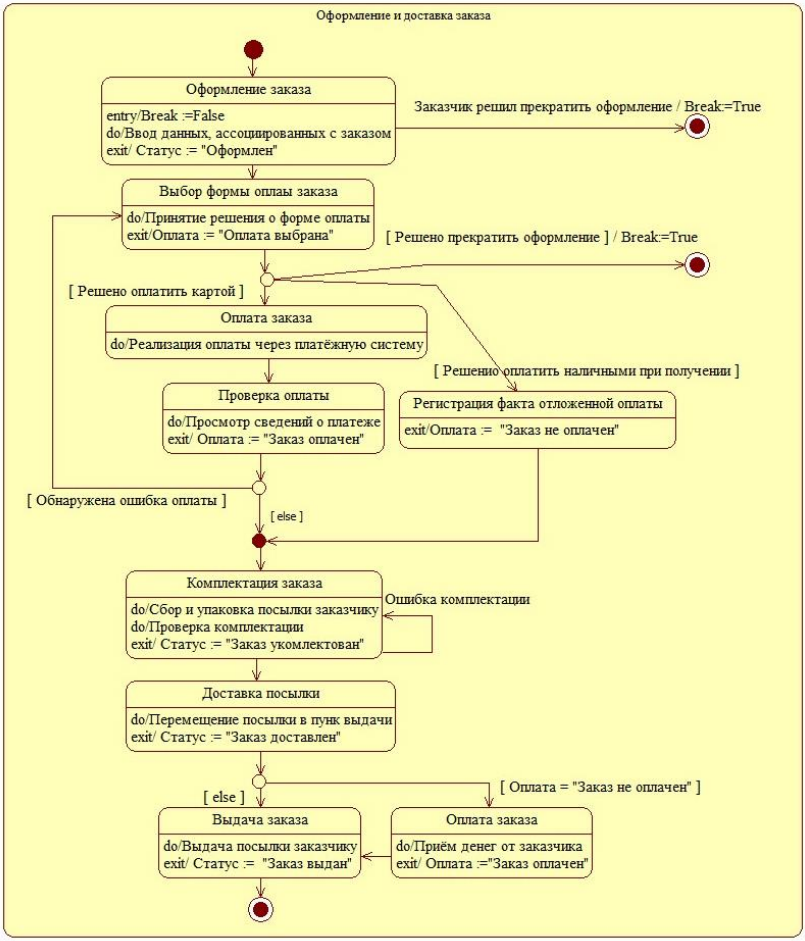


Рисунок 3.5 – Алгоритм реализации прецедента «Оформление и доставка заказа» в форме автомата

4 ИЗУЧАЕМАЯ ТЕМА «ДИАГРАММА КЛАССОВ»

Предполагается, что учащиеся знакомы с основными понятиями объектно-ориентрованного программирования.

4.1 Ассоциированные понятия

Класс. Объект класса. Роль класса. Члены класса: атрибуты, операции и их признаки. Создание класса в проекте. Определение признаков членов класса.

4.2 Руководящие принципы построения диаграммы классов

4.2.1 Общие замечания. Выделение классов

Классы участвуют в представлении всех основных аспектов рассмотрения ИС: 1) операции (методы) классов представляют функциональность системы; 2) атрибуты (поля) классов представляют информационность системы; 3) объекты/роли классов, участвующие в диаграммах последовательности (коммуникации), отражают поведение ИС; 4) классы, в своей совокупности, участвуя в модели исходного кода на диаграммах компонентов, описывают структуру системы.

Диаграммы классов целесообразно создавать на основе диаграмм поведения. Основные из которых: Use Case Diagram (текстовое описание потоков поведения), Activity Diagram, State Machine (State Chart) Diagram, Sequence Diagram: для отдельных прецедентов, состояний и деятельностей, создаются свои классы, операции (методы) которых и реализуют эти прецеденты, состояния и деятельности. Например, рассматривая алгоритм реализации сценария «Оформление заказа» (рисунок 2.3), можно представить фрагмент системы, участвующей в реализации этого сценария и связанный на диаграмме деятельности с дорожкой «Система», в виде одного – обобщённого класса с именем «Система». Операции этого класса представляют собой действия, которые необходимо выполнить, реализуя сценарий (рисунок 4.1). На диаграмме деятельности действия системы, подлежащие выполнению, представляются элементами ActionState – состояниями действий. При переходе к представлению системы в виде обобщённого класса, состояния действия трансформируются в операции класса. Например, действие «Отображение формы с опциями» превратилось в одноимённую операцию «Отображение формы с опциями()», действие «Отображение формы для ввода атрибутов заказа» превратилось в одноимённую операцию «Отображение формы для ввода атрибутов заказа()» и т. д.

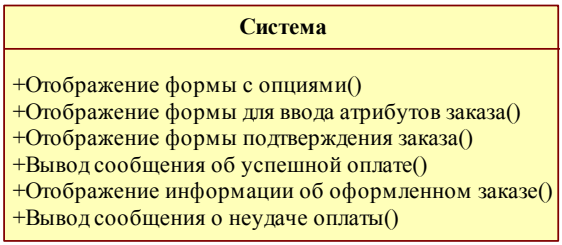


Рисунок 4.1 – Представление системы в виде одного класса с основными операциями

Декомпозиция этого обобщенного класса приводит к получению следующего набора классов:

a) граничные (boundary):
   1) форма с опциями;
   2) форма ввода атрибутов;
   3) форма подтверждения заказа;
   4) форма информации о заказе;
   5) Успех оплаты;
   6) Неудача оплаты;
b) управляющие (control):
   7) Controller;
   8) Обновление БД;
c) сущности (entity):
   9) ИнфоЗаказа.

Формы 1 – 6 реализуют соответствующие диалоги с покупателем, и могут реализовываться в виде обычных форм системы программирования. Классы 5 и 6 – это классы типа MessageBox, предназначенные для вывода кратких сообщений пользователю.

Класс 7 предназначен для локализации управления последовательностью реализуемых действий – главная форма в процессе реализации сценария, может не отображаться в виде окна, но реализовываться в виде обычной формы. Класс 8 реализует взаимодействие с базой данных, может реализоваться в виде обычной формы, неотображаемой во время реализации сценария.

Класс 9 – структура данных для хранения информации, ассоциированной с заказом, включая сведения о покупателе, выбранные им товары и результаты диалога с покупателем во время оформления заказа. Может реализоваться в виде обычной формы, неотображаемой во время реализации сценария.

Графическое представление выделенных классов показано на рисунке 4.2.

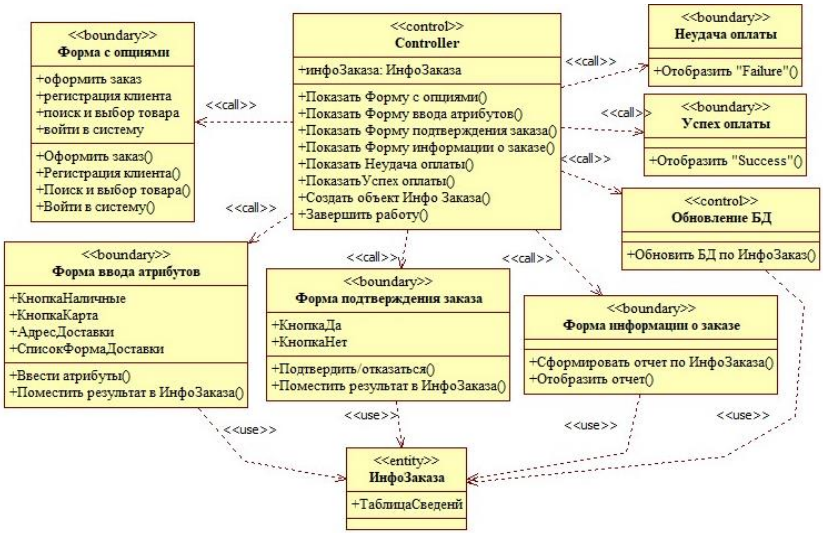


Рисунок 4.2 – Классы реализации сценария «Оформление заказа» с отношениями

Заметим, что формирование обобщённого класса «Система» носит, главным образом, учебно-методический характер – это сделано для того, чтобы показать трансформацию действий, выполняемых системой в рамках диаграммы деятельности (рисунок 2.3), в операции (методы) класса. Учащиеся призваны осознать факт сохранения поведения системы в процессе выделения классов: действия не исчезают, но приобретают форму операций. После уяснения этого факта учащиеся могут выделять классы системы по диаграмме деятельности, минуя обобщённый класс, – ставя классы в соответствие действиям, используя ориентировочный принцип:

каждому действию (ActionState) свой класс.
The ActionState –> The Class.

Однако, это принцип весьма ориентировочен – иногда некоторой группе действий может оказаться достаточным одного класса. Ситуация, когда для одного действия (ActionState) может потребоваться два или более классов будет невозможна, если в процессе выявления классов двигаться слитно по содержимому всех диаграмм, включая декомпозиционные, – на которые осуществляется ссылка инструментом SubactivityState. Элементы же ActionState, должны представлять собой действия, алгоритмы реализации которых, читателю известны, и остаётся только прописать соответствующий программный код.

Обратим внимание на то, что термин «Система» на диаграмме рисунка 2.3 символизирует не всю систему в целом, а её отдельный фрагмент, – участвующий в процессе реализации одного прецедента – «Оформление заказа». Очень часто выделение классов целесообразно начинать со сценария использования системы актором. В общем случае этот сценарий оперирует с несколькими вариантами использования, например, как это показано на рисунке 4.3.

Рисунок 4.3 – Варианты использования системы клиентом
(фрагмент диаграммы прецедентов)

Сценарий использования системы клиентом, согласно предлагаемой методике, следует представить диаграммами деятельности или автомата, и по ним уже выделять классы. При использовании диаграмм деятельности классы ставятся в соответствие состояниям действия (ActionState), а при использовании диаграмм автомата классы ставятся в соответствие простым состояниям, представляемым инструментом (State) по следующей схеме:

каждому простому состоянию (State) свой класс. *The State –> The Class.*

Автоматное представление алгоритмической реализации фрагмента, указанного на рисунке 4.3, изложено в теме «Диаграмма автомата». Там на рисунке 3.1 представлена диаграмма сценария первого уровня. На ней три простых состояния:

1) отображение интерфейсного окна и ожидание выбора клиента;
2) принятие решения о входе;
3) принятие решения о выборе товаров.

И три ссылочных состояния:

1) регистрация клиента;
2) вход в систему;
3) поиск и выбор товара.

Согласно изложенному выше правилу, ставим простым состояниям классы. При этом только один класс – «Отображение интерфейсного окна и ожидание выбора клиента» – оказывается программно реализуемым, а два других класса – «Принятие решения о входе» и «Принятие решения о выборе товаров» реализуются «условным» классом – клиентом.

Для продолжения процесса выделения классов следует перейти к реализациям (декомпозиции) ссылочных состояний. Для одного из них – «Регистрация клиента» – в теме «Диаграмма автомата» три версии сценария. Версия 1 определяет два класса – «Отображение интерфейсного окна» и «Действия клиента». Версия 2 определяет четыре класса, а версия 3 – шесть классов – по числу простых состояний.

Указанные правила следует применять и для выделения классов, реализующих остальные варианты использования.

Итак, схема выделения классов для реализации вариантов использования такова:

1. Разрабатывается диаграмма вариантов использования до уровня прецедентов «с очевидной реализацией».
2. Осуществляется алгоритмическая реализация прецедентов:
   - либо в виде диаграммы деятельности с декомпозицией до уровня состояний действия (алгоритм их реализации считается очевидным, и далее можно осуществлять уже программную реализацию);
   - либо в виде диаграммы автомата с декомпозицией до уровня простых состояний (алгоритм их реализации так же считается очевидным).
3. Каждому состоянию действия / простому состоянию ставится в соответствие свой класс.

4.2.2 Представления классов

Классы разделяются по пакетам: 1) классы-сущности (носители информации); 2) граничные классы (формы пользовательского интерфейса); 3) управляющие классы (решатели прикладных задач, т. е. исполнители процессов).

Диаграмму классов следует представлять в трёх видах:

1) диаграмма пакетов с вложенными в пакеты классами; классы в этих диаграммах можно (хотя и необязательно) представлять кратко – одним именем;

2) диаграмма детального представления классов; классы в этих диаграммах следует (обязательно) представлять максимально подробно – помимо имён должны указываться все атрибуты и все операции с указанием всех их признаков – спецификаторов доступа (+ | - | # | □), типов

атрибутов, типов параметров операций и типов результатов, возвращаемых операциями; при построении этой версии диаграммы следует тщательно задокументировать все члены класса, – указать их назначение и обосновать их атрибуты и типы;

3) диаграмма отношений между классами: на этих диаграммах степень детальности представления классов может быть произвольной и *разной* для *разных* классов, – для одних классов можно приводить только атрибуты, отменив представление методов, для других наоборот – можно указать только методы; а для третьих – можно ограничиться одними именами классов; самое главное для этой версии диаграммы – *детальное документирование* стрелок, выражающих отношения между классами. Примером представления отношений между классами является рисунок 4.2.

Отображая отношения между классами, следует понимать следующее.

Отношения ассоциации

Отношения ассоциации имеют три разновидности: 1) собственно ассоциации; 2) агрегации и 3) композиции. Их можно назвать отношениями по данным, поскольку все они на диаграммах проявляются спецификой атрибутов.

Собственно ассоциация соответствует ситуации связи таблиц базы данных с использованием внешних ключей – в классах имеются ссылочные атрибуты с одинаковым смыслом и одинаковыми значениям, при этом один из классов может быть категорирован как зависимый, а другой как независимый: ссылочный атрибут первого класса заполняется ранее определённым значением ссылочного атрибута второго класса. В диаграммах классов стрелка агрегации направляется от зависимого класса к независимому. Иногда говорят о навигации – имея доступ к объекту зависимого класса, можно отыскать объект независимого класса. В обратную сторону такая навигация может быть невозможной, – если в объекте независимого класса нет информации о ключевом атрибуте зависимого класса. В диаграммах стрелки ассоциаций целесообразно снабжать именами – аналогами глагольных фраз (verb phrase) диаграмм ERD.

Агрегация и композиция соответствуют отношению «часть-целое», и в диаграммах классов проявляются в том, что типом одного из атрибутов некоторого класса (представляющего «целое») является другой класс (представляющий «часть»), Во время исполнения (Runtime) этот атрибут будет содержать ссылку на объект другого класса.

Концептуально агрегация и композиция отличаются временем жизни «частей»: в случае агрегации «части» продолжают существовать

после удаления «целого», а в случае композиции «части» удаляются вместе с «целым».

Отличия агрегации и композиции в диаграммах заключаются только в изображении ромбика на конце стрелки отношения, упирающейся в класс-целое, – в случае агрегации он не закрашен, а в случае композиции закрашен. Выбор отношения модельером должен быть обоснован в документировании стрелки отношения.

На этапе реализации проектируемой системы агрегация и композиция реализуются соответствующим программным кодом деструкторов «целого» – он либо не включает (в случае агрегации), либо включает (в случае композиции) обращения к деструкторам «частей».

Отношения зависимости

Весьма часто классы находятся друг с другом в отношении зависимости (Dependency). Эти отношения отображаются штриховой линией, направленной от зависимого класса (Client в терминологии стандарта языка UML) к независимому (Supplier). Стрелка может снабжаться стереотипом или произвольным названием. Набор стандартных стереотипов зависимости представлен в таблице 9 [3, с. 98], содержащей, к сожалению, ошибку в пояснении стереотипа «`instantiate`» (создавать экземпляр). Содержится; «Указывает, что операции независимого класса создают экземпляры зависимого класса». Должно быть наоборот: «A stereotyped usage dependency among classifiers indicating that operations on the client create instances of the supplier», или по-русски: «Стереотипная зависимость использования между классификаторами, указывающая на то, что операции зависимого класса (клиента) создают экземпляры независимого класса (поставщика)».

Заметим, что в указанной разновидности зависимости вопрос о том, кто является поставщиком, а кто клиентом иногда нетривиален. Например, в отношении классов «Автозавод» и «Автомобиль» [5, с. 39] поставщиком (независимым классом) является «Автомобиль», а клиентом – «Автозавод». На первый взгляд, кажется, что должно быть наоборот, – завод важнее автомобиля, он производит автомобили, значит, он независим, а автомобиль зависит от его производителя. Но дело в следующем: «Автомобиль» в информационном плане – это конструкторская документация, которая используется (потребляется) заводом, информационным обликом которого (в рассматриваемом отношении) является производственно-технологическая документация, создаваемая на основе конструкторской документации. Изначально создаётся конструкция автомобиля, а уже затем она используется для создания технологии воплощения этой конструкции в автомобиль. Таким образом,

«Автомобиль» поставляет «Автозаводу» свою конструкцию, а последний создаёт на основе конструкции технологию её материализации.

В практике проектирования систем часто возникает вопрос о выборе вида отношения между классами – ассоциация или зависимость? При этом полезно иметь в виду следующее: 1) ассоциации (все разновидности) предназначены для отображения связей между классами по данным – объекты ассоциируемых классов объединяются для того, чтобы сформировать информационное пространство, необходимое для решения некоторой задачи; естественно, чаще всего, ассоциируются классы-сущности, – объекты которых предназначены для хранения информации; 2) зависимости предназначены для отображения:

- связей между процедурными классами, объекты которых предназначены для вызова методов обработки информации (эти зависимости целесообразно изображать со стереотипом «`call`»),
- связей процедурных классов с классами-сущностями (их целесообразно изображать со стереотипом «`use`»).

Отношения зависимости используются также для отображения связей объектов и классов по происхождению:

- «`instanceOf`» – объект (зависимый элемент модели) является экземпляром некоторого класса (независимого элемента модели); создание данного объекта осуществляется объектом другого класса;
- «`instantiate`» – объект зависимого класса создаёт объект независимого класса; типично для реальной практики программирования – метод клиента создаёт объект поставщика для получения доступа к необходимой функциональности;
- «`powertype`» – полная аналогия «`instantiate`», но на уровне классов – независимый класс создаётся зависимым метаклассом;
- «`refine`» – зависимый класс уточняет независимый класс;
- «`derive`» – зависимый класс может быть восстановлен (получен) из независимого класса.

Может также возникнуть вопрос: могут ли классы находиться одновременно и в отношении ассоциации и зависимости? Ответ положительный – могут. Например, класс «Факультет» может иметь в качестве атрибутов названия должностей сотрудников и одновременно может иметь операцию «ПринятьНаДолжность(Персона:string)», которая вызывает операцию «Оформить(Персона:string)» класса «Должность». Таким образом, «Факультет» и «Должность» связаны композицией и зависимостью «Факультета» от «Должности».

5 ИЗУЧАЕМАЯ ТЕМА «ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ»

5.1 Ассоциированные понятия

Sequence Diagram = Диаграмма последовательности

Communication Diagram = Collaboration Diagram = Диаграмма коммуникации / кооперации (сотрудничества)

D3 = D4 – диаграммы последовательности и коммуникации семантически эквивалентны.

Combined Fragment = комбинированный фрагмент, – средство для представления управляющих структур (следования, ветвления, циклов, параллельностей), контейнер для размещения операндов взаимодействия.

Interaction Operand = операнд взаимодействия – средство для размещения последовательностей действий, к которым применяются правила управления, определяемые оператором взаимодействия.

InteractionOperator = оператор взаимодействия – свойство комбинированного фрагмента, определяющее тип управления для последовательностей, помещаемых в Interaction Operand комбинированного фрагмента. Примеры операторов взаимодействия:

a) alt – ветвление на произвольное количество направлений, – аналог оператора switch в программировании;
b) opt – выполнение по условию, аналог сокращённой формы оператора if – if then;
c) loop – цикл типа while (цикл с предусловием), условие цикла – это условие выполнения тела цикла – содержимого Interaction Operand комбинированного фрагмента; стандарт UML предусматривает использование loop и как оператора for, но в StarUML это не реализуемо – в нём нет средств для указания границ значений счетчика циклов;
d) strict – строго последовательно, – так выполняются процессы, когда нет никаких комбинированных фрагментов;
e) par – процессы, размещённые в Interaction Operand, выполняются параллельно;
f) seq – слабая последовательность (Weak Sequencing) означает, что последовательности, выполняемые разными объектами, выполняются параллельно, а последовательности, выполняемыми одним объектом, выполняются последовательно друг за другом в порядке их размещения по вертикали диаграммы.

5.2 Руководящие принципы построения диаграммы

5.2.1 Общие замечания

Диаграмма последовательности – одно из альтернативных средств графического представления поведения системы. Во-первых, её абсолютным достоинством является строгая структурированность представляемого сценария – Interaction Operands всех Combined Fragments наглядно включают другие управляющие структуры, исключая возможность пересечения структур, т. е. нарушения структурированности моделируемых процессов. Во-вторых, весьма важно и то, что диаграмма последовательности сочетает в себе представление алгоритмичности процесса с представлением структуры системы: алгоритм представляется последовательностью стимулов (сообщений), а структура – объектами-обработчиками стимулов. Нечто подобное представимо и диаграммой деятельности, – если в ней изобразить объекты системы в виде названий плавательных дорожек.

5.2.2 Порядок построения диаграмм последовательности

Хотя диаграммы последовательности и являются равноправными представителями средств моделирования поведения систем, у них есть всё же важная особенность – для их построения целесообразно предварительно выделить классы, реализующие моделируемое поведение. Для выделения классов полезно построить либо диаграмму деятельности, либо диаграмму автомата. Таким образом, имеет место следующая цепочка построений (рисунок 5.1):

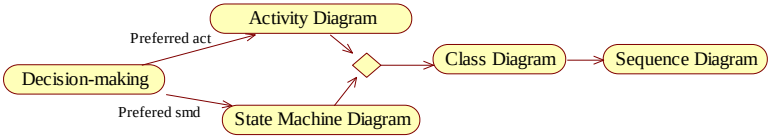


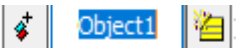
Рисунок 5.1 – Типовая схема построения диаграмм последовательности

В то же время, если алгоритм реализации процесса несложен, то вполне возможно обратное движение (рисунок 5.2).

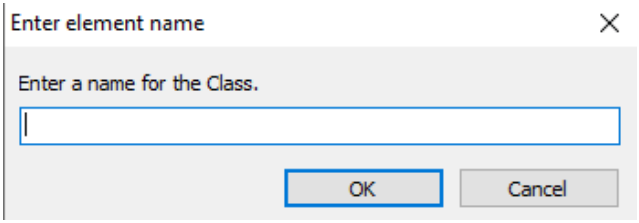
Рисунок 5.2 – Инверсная схема построения
диаграмм последовательности

При этом на диаграмму последовательности изначально наносятся нетипизированные объекты количеством равным ожидаемому коли-

честву операций. Затем объектам назначаются классы следующими манипуляциями.

1. Double click на объекте. // Появляется диалоговое окно: 

.
2. Нажимается правая пиктограмма. // Появляется диалоговое окно:



3. Вводится имя класса. Нажимается кнопка «OK» на окне.

Класс создаётся в проекте и отображается в инспекторе модели на уровне Logical View.

После чего на диаграмму наносятся стрелки стимулов (сообщений), для которых назначаются вызываемые операции следующими манипуляциями.

1. Double click по стрелке // Появляется диалоговое окно: 
2. Вводится имя операции. Нажимается клавиша Enter.

Введённое имя немедленно становится именем операции класса, что и отображается в инспекторе модели на уровне имени класса. Созданные таким образом классы могут использоваться в любых диаграммах классов.

Следует строго помнить: надписи над сплошными стрелками – это вызовы операции (могут быть также сигналами, указаниями создания или ликвидации объектов) тех классов, в линию жизни объектов которых втыкается стрелка. Поэтому эти операции обязательно должны присутствовать на диаграмме классов.

5.2.3 Примеры диаграмм последовательности

Пример диаграммы последовательности показан на рисунке 5.3. Обратим внимание на особый статус объекта класса `Controller`: он задаёт порядок выполнения операций, вызываемых через объекты других классов, т. е. он является формирователем моделируемого сценария, укладывая в необходимую последовательность операции граничных классов, `Платёжной системы` и класса `Обновление БД`.

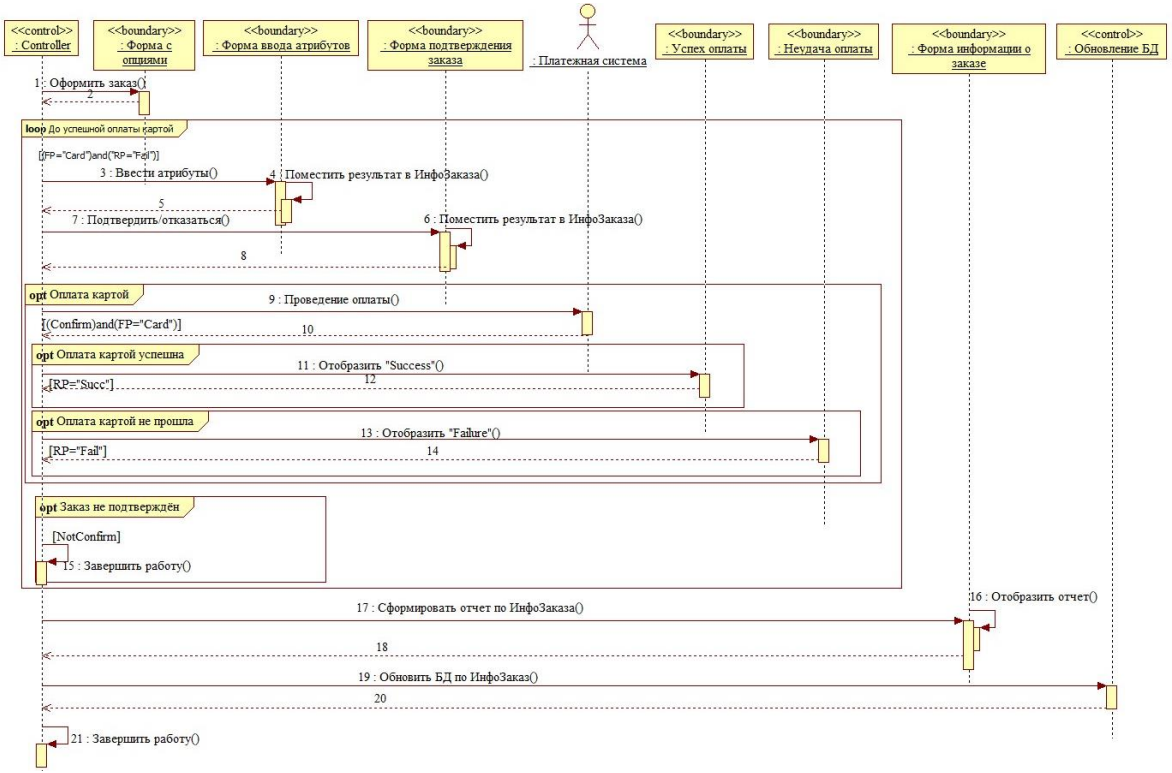


Рисунок 5.3 – Сценарий «Оформление заказа» в виде диаграммы последовательности

ПРИЛОЖЕНИЕ А
Шаблон титульного листа отчета по теме

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.Ф. УТКИНА
(ФГБОУ ВО "РГРТУ", РГРТУ)

Кафедра вычислительной и прикладной математики

Отчет по теме
«**ДИАГРАММА ПРЕЦЕДЕНТОВ**»
по дисциплине «Проектирование информационных систем»
(или «Проектирование программных систем»»

Выполнил: ст. гр. _____

_____

Проверил: _____

_____

Рязань 202__ г.

ПРИЛОЖЕНИЕ Б
Темы заданий к лабораторным и практическим работам

1. Проектирование системы интернет-бронирования гостиницы.
2. Проектирование системы реализации готовой продукции.
3. Проектирование системы интернет-заказов товаров магазина электроники.
4. Проектирование системы предоставления и запроса вакансий для бюро по трудоустройству.
5. Проектирование системы электронной записи клиентов нотариальной конторы.
6. Проектирование системы интернет-заказов у поставщиков автозапчастей.
7. Проектирование системы записи и учета прохождения курсов повышения квалификации.
8. Проектирование электронной системы учета оценок студентов.
9. Проектирование электронной системы распределения нагрузки преподавателей.
10. Проектирование информационной системы страховой компании.
11. Проектирование системы контроля сроков и обслуживания клиентов ломбарда.
12. Проектирование электронной системы записи на прием пациентов частной клиники.
13. Проектирование системы учета кадров на предприятии.
14. Проектирование электронной системы заказа книг в библиотеке.
15. Проектирование театральной интернет-кассы.
16. Проектирование системы бронирования для проката автомобилей.
17. Проектирование системы учета рекламы в эфире телеканала.
18. Проектирование системы электронного расписания работы телеканала.
19. Проектирование системы интернет-заказов ювелирной мастерской.
20. Проектирование интернет-магазина одежды.
21. Проектирование электронной системы сдачи в аренду торговых площадей.
22. Проектирование системы продажи и бронирования билетов кинотеатра через интернет.

23. Проектирование интернет-афиши и справки кинотеатра.

24. Проектирование системы учета технического обслуживания станков.

25. Проектирование информационной системы турфирмы.

26. Проектирование системы покупки и бронирования билетов на поезд.

27. Проектирование информационной системы компании грузоперевозок.

28. Проектирование системы учета телефонных разговоров сотрудников.

29. Проектирование интернет-системы подачи заявок на оформление кредита.

30. Проектирование интернет-кабинета клиента банка.

31. Проектирование информационной системы агенства недвижимости.

32. Проектирование интернет-системы записи и учета скидок клиентов салона красоты.

33. Проектирование системы регистрации и контроля сообщений участников интернет-форума.

34. Проектирование системы доставки товаров из магазина.

35. Проектирование интернет-системы заказа и доставки пиццы.

36. Проектирование информационной системы детского сада.

37. Проектирование системы курсов дистанционного обучения.

38. Проектирование системы футбольных ставок.

39. Проектирование системы бронирования столиков и заказа блюд меню ресторана по интернету.

40. Проектирование системы обслуживания клиентов частной почтовой службы.

41. Проектирование системы учета сбыта продукции сельскохозяйственного предприятия.

42. Проектирование системы маркетинга предприятия.

43. Проектирование информационной системы компании прямых продаж косметики.

44. Проектирование каталога и системы заказов легковых автомобилей по интернету.

45. Проектирование системы гарантийного обслуживания электротоваров.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Белов В.В., Чистякова В.И. Проектирование информационных систем: учебник – М. : КУРС, 2018. – 400 с. ISBN 978-5-906923-53-0 (КУРС) (45 экз. в БФ РГРТУ).

2. Белов В.В., Чистякова В.И. Проектирование информационных систем: учебник для студ. учреждений высш. образования / Под ред. В.В. Белова. – 2-е изд., стер. – М.: Издательский центр «Академия, 2015. – 352 с. (Сер. Бакалавриат). ISBN 978-5-4468-2440-3 (132 экз. в БФ РГРТУ)

3. Иванов, Денис Юрьевич. Унифицированный язык моделирования UML [Электронный ресурс]: учебное пособие для вузов по направлению подготовки "Системный анализ и управление" / Д.Ю. Иванов, Ф.А. Новиков; Санкт-Петербургский государственный политехн. ун-т. – Электрон. текстовые дан. (1 файл: 1,83 Мб). – Санкт-Петербург, 2011. – Загл. с титул. экрана. – Электронная версия печатной публикации. – Свободный доступ из сети Интернет (чтение, печать, копирование). – Текстовый документ. – Adobe Acrobat Reader 7.0. Доступно по URL:http://elib.spbstu.ru/dl/2962.pdf, http://elib.spbstu.ru/dl/2962.pdf/download/2962.pdf

4. Каюмова А.В. Визуальное моделирование систем в StarUML: Учебное пособие/ А.В. Каюмова. Казань. – Казанский федеральный университет, 2013. – 104 с.

5. OMG® Unified Modeling Language® (OMG UML®). Version 2.5.1// OMG Document Number: formal/2017-12-05. Normative URL: https/www.omg.org/spec/UML/ – 796 pp.

6. Орлов С.А. Программная инженерия. Технологии разработки программного обеспечения: Учебник для вузов. –5-е изд. обновл. и доп. Стандарт третьего поколения. – СПб.: Питер, 2016. – 640 с. Электрон. текстовые дан. (1 файл : 37,58 Мб). — Текстовый документ. — Adobe Acrobat Reader, Internet Explorer. Доступно по URL https://www.twirpx.com/file/2378219/.

7. Оформление текстовых алгоритмов: методические указания к лабораторным работам и практическим занятиям / Рязан. гос. радиотехн. ун-т. им. В.Ф.Уткина; Сост.: В.В. Белов, В.И. Чистякова. Рязань, 2022. 24 с. [Номер в библиотеке РГРТУ: 7466]