Владимирский государственный университет

ПРОГРАММИРОВАНИЕ ДЛЯ MICROSOFT .NET

Методические указания к лабораторным работам
по дисциплине «Технологии программирование»
Часть 3

Владимир 2013

ПРЕДИСЛОВИЕ

Необходимость и важность обеспечения высокого уровня разрабатываемых программных систем обуславливает использование современных технологий и подходов к их проектированию. Такой подход позволяет существенно повысить уровень программных разработок.

Использование новых технологий возможно только при условии их полного изучения и владения ими. Поэтому изучение существующих подходов позволяет сделать существенный прорыв в индустрии программного обеспечения и в подходах разработки программ.

Предлагаемый в данных методических указаниях к лабораторным работам по курсу «Программирование» материал ставит целью познакомить студентов с различными аспектами программной инженерии. С одно стороны это изучение процедур анализа и проектирования программных систем, с другой стороны их реализация на базе выбранных средств и технологий.

Первый цикл работ (с первой по третью) дает совокупное представление о построении взаимоотношений между программной системой и элементами внешней среды и позволяет научиться определять границы исследуемой системы и выделять её функции в виде прецедентов. Последующие работы этого же цикла знакомят с методами и средствами построения моделей для различных уровней проектирования ИС с использованием графического языка моделирования UML и средства Microsoft Visio.

Второй цикл работ (с четвертой по седьмую) дает основы WEB-программирования. Знания, полученные в этой части работ, позволяют освоить подходы разработки как простых WEB-приложений, так и более серьезных приложений, ориентированных на обработку и сохранение данных в базах данных.

Лабораторная работа №1

АНАЛИЗ ПРЕЦЕДЕНТОВ РАБОТЫ С ПРОГРАММНОЙ СИСТЕМОЙ. МОДЕЛИРОВАНИЕ ВЗАИМООТНОШЕНИЙ ИС И ЭЛЕМЕНТОВ ВНЕШНЕЙ СРЕДЫ. РАЗРАБОТКА ДИАГРАММ ПРЕЦЕДЕНТОВ

1. Цель работы

Научиться моделировать взаимоотношения элементов внешней среды с элементами проектируемой программной системы через синтез прецедентов, их расширенное описание и включение диаграмму UseCase.

2. Общие сведения

Никакие системы не существуют в изоляции. Как правило, они взаимодействуют с элементами внешней по отношению к ним (участниками взаимодействия, актерам) – людьми или программами – которые используют систему в своих целях, причем каждый актер (actor) ожидает, что она будет вести себя определенным, вполне предсказуемым образом.

Диаграммы прецедентов позволяют визуализировать поведение системы, подсистемы или класса, чтобы пользователи могли понять, как их использовать, а разработчики - реализовать соответствующий элемент.

Прецеденты состоят из множества сценариев (последовательности шагов описывающих взаимодействие между пользователем и системой), объединенных вместе некоторой общей целью пользователя. Относительно сложная система содержит несколько десятков прецедентов, каждый из которых может разворачиваться в несколько десятков сценариев. Для любого прецедента можно выделить основные сценарии, описывающие важнейшие последовательности, и вспомогательные, описывающие альтернативные последовательности. У каждого прецедента есть предусловие и постусловие.

В качестве примера объекта информатизации в рамках настоящего лабораторного практикума мы выберем библиотеку. В данном случае актером, элементом, взаимодействующим с будущей ИС "Библиотека" будет читатель. Как правило, у читателя, зашедшего в библиотеку, есть два сценария поведения – поиск и выбор нужной ему литературы и передача прочитанных им книг обратно в распоряжение библиотеки.

В нашем случае мы рассмотрим прецедент "Выбирать литературу" который охватывает два подсценария: успешного нахождения нужной литературы и неудачного поиска. Выбранный прецедент подробно описан в таблице 1.

Таблица 1

Расширенное описание прецедента «Выбирать литературу»

Название: «Выбирать литературу» **Предусловие:** читатель зарегистрирован в ИС "Библиотека", т.е. имеет присвоенный ему номер читательского билета, он авторизован в системе. **Действующее лицо:** читатель **Основной поток**: *Выбирать литературу* Читатель открывает форму (окно приложения), отображающую каталог литературы. Пользуется средствами поиска по каталогу (вводит автора или название издания в соотв. поля). Отмечает необходимую литературу и нажимает кнопку "Выписать требование" - данные найденной книги или журнала (прецедент "Выписать требование"). Система сохраняет требование в БД и перенаправляет его библиотекарю. **Альтернативный поток**: книга в каталоге отсутствует. На 2 шаге читатель не находит требующуюся книгу или журнал. В этом случае он: либо начинает поиск другой литературы, либо корректирует условия поиска. **Постусловие:** если книга или журнал найдены должно быть выписано требование на литературу – запрос на выдачу найденной литературы.

Стоит заметить, что словесное описание любого прецедента, даже самого элементарного, должно включать в себя как минимум 3 раздела – предварительное условие, основной поток, постусловие. Однако, в общем случае описание прецедента может включать в себя один или несколько альтернативных потоков, связанных с различными шагами основного потока, ссылки на другие прецеденты, указание конкретных актеров, вовлеченных в прецедент (раздел "Действующие лица/актеры"). Последнее связано с тем, что прецедент могут инициировать несколько актеров, и в зависимости от конкретного актера прецедент может включать в себя специфические шаги основного потока или даже специфические альтернативные потоки.

После словесного описания **каждого** из сценариев прецедентов можно приступить к построению диаграммы прецедентов (Use case diagram). На ней показывается совокупность прецедентов и актеров, а также отношения между ними.

Рассмотрим основные элементы диаграммы прецедентов. На диаграмме ниже представлены упомянутые выше прецеденты "Choose" и "Write Requirement".

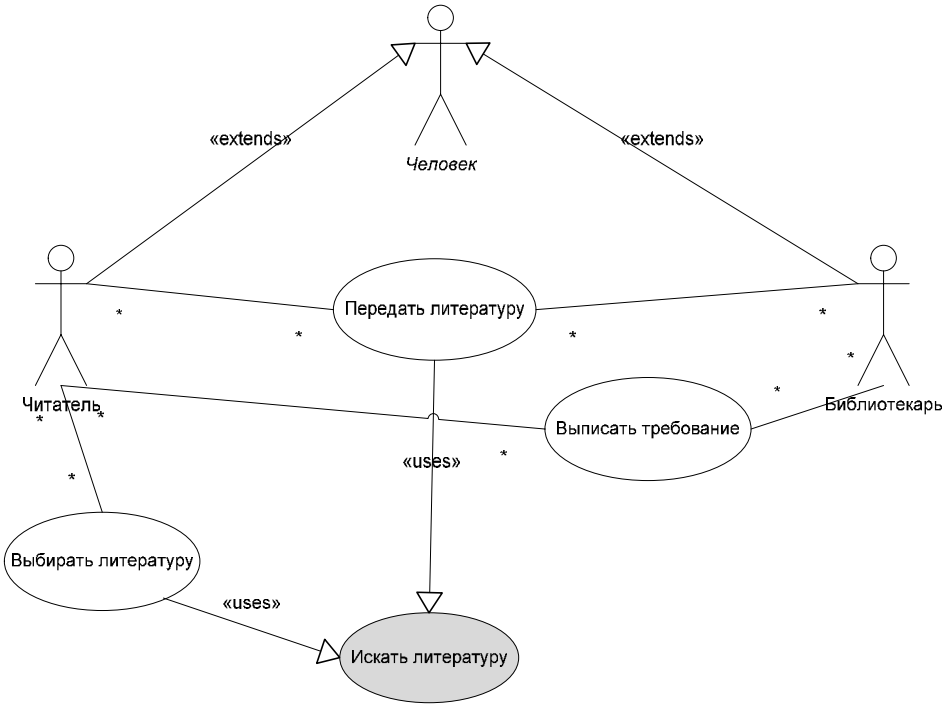

Рисунок 1. Диаграмма прецедентов

Графически прецеденты изображаются в виде эллипсов. Прецеденты бывают двух типов: бизнес-прецеденты (описывают функциональность на верхнем уровне и предназначена для заказчика программной системы) и системные прецеденты (описываю функциональность на нижнем уровне, строятся для разработчика программной системы).

У любого прецедента должно быть имя, уникальное в рамках пакета. Имя может занимать несколько строк. На практике для именования прецедентов используют короткие глагольные фразы в активной форме, обозначающие некоторое поведение Прецеденты можно организовать, определив между ними отношения обобщения, расширения и использования.

Отношение *обобщения* между прецедентами аналогично отношениям обобщения между классами. Это означает, что прецедент-потомок наследует поведение и семантику своего родителя, может замещать его или дополнять его поведение, а кроме того, может быть подставлен всюду, где появляется его родитель. Отношение обобщения следует использовать при описании изменения некоторого нормального поведения.

Отношение *использования* (uses) имеет место, когда существует какой-либо фрагмент поведения системы, который повторяется более чем в одном прецеденте. Например, для обоих прецедентов "Выбрать литературу" и "Искать литературу" системе, равно как и ее пользователям – библиотекарю и читателю – необходимо найти нужную литературу. Описание

поиска журналов или книг можно представить как отдельный сценарий. Поэтому выделяем отдельный прецедент "Искать литературу".

Отношение *расширения* (extend) по своей сути аналогично отношению обобщения. При построении модели расширяющий прецедент может дополнять поведение базового прецедента, но в последнем должны быть определены точки расширения (только для UML). При этом расширяющий прецедент может дополнять поведение базового только в этих точках расширения. Отношение расширения используется при более точном описании изменения некоторого нормального поведения.

На рисунке 2, Б представлен пример применения отношения расширения для фрагмента предметной области "ИС супермаркета". Прецедент "Оплатить покупку с дисконтной картой" расширяет прецедент "Оплатить покупку" дополнительными действиями – вводом данных о дисконтной карте; точка расширения – момент, когда необходимо ввести информацию о карте – указано в прецеденте "Оплатить покупку".

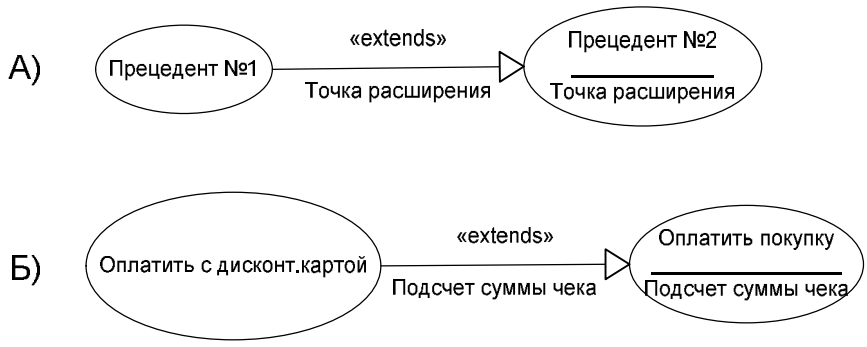

Рисунок 2. Отношение расширения (А - общая схема, Б – пример отношения расширения)

Актер представляет собой некоторую роль, которую играет пользователь системы. Графически актеры изображаются в виде "человечков" (рис. 3). Актеры не обязательно должны быть людьми. В качестве актера может выступать любой внешний объект, пользующийся информацией от разрабатываемой системы.

Рисунок 3. Графическое изображение актера

На рис. 1 показаны три актера: "Человек", "Библиотекарь" и "Читатель". Между актерами так же могут возникать различные отношения

(обобщение, агрегация, зависимость). Например, "Читатель" и "Библиотекарь" обобщаются актером "Человек".

В библиотеке может быть много читателей и библиотекарей, но так как по отношению к системе они играют одни и те же роли, нам нет необходимости рассматривать конкретных читателей и библиотекарей в отдельности. Один и тот же человек может несколько ролей. Например, библиотекарь тоже может взять почитать книгу.

Актер может выполнять несколько прецедентов, и соответственно прецедент может выполнять несколько актеров.

Диаграммы прецедентов имеют большое значение для визуализации, специфицирования и документирования поведения элемента. Они облегчают понимание систем, подсистем или классов, представляя взгляд извне на то, как данные элементы могут быть использованы в соответствующем контексте. Кроме того, такие диаграммы важны для тестирования исполняемых систем и для понимания их внутреннего устройства.

3. Порядок выполнения работы

1 .Перед выполнением предлагаемых лабораторных работ рекомендуется ознакомиться со следующими дополнительными источниками:

- Леоненков А.В. Нотация и семантика языка UML (лекции №1-9) http://www.intuit.ru/department/pl/umlbasics/
- Грекул В.И. Проектирование информационных систем (лекции №1-3, 11, 12) http://www.intuit.ru/department/se/devis/

2. Для выполнения данной лабораторной работы здесь и далее необходимо программное средство Mcrosoft Office Visio 2003 и более новые версии (для выполнения практикума потребуется набор графических шаблонов "UML Model Diagram (Metric Units)")

3. Выбрать и подробно исследовать объект информатизации . Кратко (порядка одного абзаца) описать выбранный ОИ. Определить границы исследуемого объекта, актеров. Описать 3-4 прецедента, по образцу, представленному в разделе 2 данной работы. Построить диаграмму прецедентов, используя в Microsoft Office Visio набор графических примитивов UML Use Case (Metric).

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Словесное описание объекта информатизации и всех выделенных прецедентов.
4. Диаграмма прецедентов.

5. Выводы по работе

5. Контрольные вопросы

1. Что такое прецедент
2. Каковы основные элементы, выносимые на диаграмму UseCase
3. Что включает в себя расширенное описание прецедента
4. Пояснить суть отношений между прецедентами (включение и расширение)

6. Варианты индивидуальных заданий

Каждый вариант индивидуального задания представляет собой описание некоторого объекта информатизации (ОИ) – предприятия, отдела или бизнес-процесса, который оперирует со специфичной информацией, характерной для данного объекта информатизации. Задача студента заключается в:

- определении границ ОИ и его взаимодействия с элементами внешней для него среды; результат – текстовое описание прецедентов и построение диаграммы прецедентов;
- определения структуры ОИ и протекающих в нем процессов обработки информации, анализе структуры ОИ и выделении атрибутов и поведения компонентов исследуемого объекта; результат – построение диаграммы классов;
- анализ поведения структурных элементов ОИ и их взаимодействия; результат – построение диаграмм деятельности.

В случае, если на этапах анализа и проектирования выяснится, что каких либо данных не хватает для логичного и законченного представления системы, студент вправе сам дополнить исследуемый объект необходимыми компонентами/атрибутами/поведением.

Итогом работы выступает сводный отчет, включающий в себя результаты работы над проектом ИС по каждому из этапов. Объекты информатизации перечислены ниже.

1. Университет.
2. Склад.
3. Производственное предприятие.
4. Сеть магазинов.
5. Сеть авторемонтных мастерских.
6. Деканат.
7. Поликлиника.
8. Телефонная станция.
9. Управление городским транспортом.

10. Авиакомпания.
11. Интернет-магазин.
12. Фермерское хозяйство.
13. Автотранспортное предприятие.
14. Отдел кадров
15. ГИБДД.

В качестве объекта информатизации студентом может быть выбран произвольный бизнес-процесс, предприятие или отдел. Работа студента в рамках лабораторного практикума оценивается по следующим принципам и критериям:

- адекватность и достоверность представления объекта информатизации в моделях; поскольку работа ведется над реально существующими объектами, перед проведением анализа выбранный ОИ должен быть подробно изучен;
- соблюдение требований методологий проектирования ИС;
- соблюдение сроков выполнения этапов лабораторного практикума.

Лабораторная работа №2

АНАЛИЗ ОБЪЕКТА ИНФОРМАТИЗАЦИИ И МОДЕЛИРОВАНИЕ СТРУКТУРЫ ПРОГРАММНОЙ СИСТЕМЫ НА ВЕРХНЕМ УРОВНЕ. РАЗРАБОТКА ДИАГРАММ КЛАССОВ

1. Цель работы

Выполнить анализ объекта информатизации и смоделировать структуры будущей программной системы на верхнем уровне в виде конечных диаграмм классов.

2. Общие сведения

Дополнительная литература к работе

Перед выполнением предлагаемых лабораторных работ рекомендуется ознакомиться со следующими источниками:

- Леоненков А.В. Нотация и семантика языка UML (лекции №1-9) http://www.intuit.ru/department/pl/umlbasics/
- Грекул В.И. Проектирование информационных систем (лекции №1-3, 11, 12) http://www.intuit.ru/department/se/devis/

  или с литературными источниками из библиографического списка методических указаний.

Необходимые программные средства

Mcrosoft Office Visio 2003 и более новые версии (для выполнения практикума потребуется набор графических шаблонов "UML Model Diagram (Metric Units)")

Моделирование структуры ИС на верхнем уровне

Моделирование предметной области позволяет сконцентрировать внимание на наиболее важных деталях проекта. Это наиболее важно при разработке крупных программных систем, когда в проект вовлечено большое число людей. Язык визуального моделирования позволяет более наглядно показать схему будущего программного продукта. Помимо этого он предоставляет возможность создавать техническую документацию и каркас кода будущего программного продукта. Основные принципы моделирования структуры ИС рассмотрим на знакомом по предыдущей лабораторной работе примере библиотеки.

У каждого читателя есть читательский билет, который однозначно идентифицирует посетителя библиотеки. Придя в библиотеку, читатель ищет по каталогу нужные ему книги и журналы. Далее читатель заполняет требование (список литературы). Далее предварительно проверив их наличие, библиотекарь выдает книги и журналы читателю.

Диаграмма классов занимает центральное место при проектировании ИС. Фактически любая методология проектирования, ориентированная на объек включает в себя некоторую разновидность диаграммы классов.

На *диаграмме классов* (Class Diagram) показывают классы, интерфейсы, объекты и кооперации, а также их отношения. Выделяют два вида отношений: ассоциации (читатель может взять книгу в библиотеке) и подтипы (библиотекарь является человеком).

Рассмотрим основные элементы диаграммы классов.

Класс (Class) - это описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Класс реализует один или несколько интерфейсов. Графически класс изображается в виде прямоугольника, в котором обычно записаны его имя, атрибуты и операции (рис. 4).

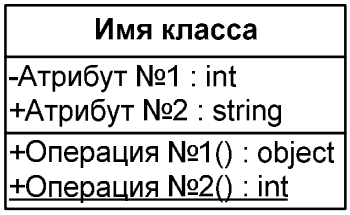

Рисунок 4. Изображение класса в UML

Ассоциация (Association) - структурное отношение, описывающее совокупность связей; связь - это соединение между объектами (рис. 4). Графически ассоциация изображается в виде прямой. Каждая ассоциация имеет два конца. Конец ассоциации может быть явно помечен некоторой меткой – ролью. Если метка отсутствует, то концу ассоциации присваивается имя целевого класса.

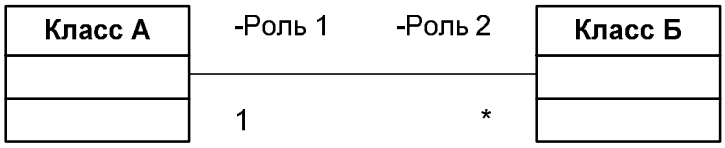

Рисунок 5. Отношение ассоциации между классами

Разновидностью ассоциации является *агрегирование* (Aggregation). В двух словах, агрегация – это когда класс в своей структуре использует часть другого класса. Например, можно сказать, что процессор или жесткий диск являются частью компьютера.

Существует более сильная разновидность агрегации – *композиция* (Composition). При композиции класс является целиком частью другого класса. Например, запись в требовании целиком является частью требования.

Конец ассоциации обладает *кратностью* (Multiplicity), которая показывает сколько объектов может участвовать в отношении. Выделяют 4 основных варианта кратности:

- "1" - только один;
- "*" - много (ноль или более);
- "0..1" - необязательная (ноль или один);
- "m..n" - заданное число.

Например, для ассоциации между Каталогом (Catalogue) и Литературным изданием (Requirement), символ "*" показывает, что в одном каталоге может (и должно) быть несколько книг, а "1" - что вся литература может находиться лишь в одном каталоге.

Существует еще одни способ связи классов – *обобщение* (Generalization) Обобщение - это отношение между общей сущностью (суперклассом, или родителем) и ее конкретным воплощением (субклассом, или потомком). Обобщения иногда называют отношениями типа "является", имея в виду, что одна сущность (например, класс Книга (Book) или Журнал (Magazine)) является частным выражением другой, более общей (скажем, класса Литературное издание (Literature)).

Класс может иметь одного или нескольких родителей или не иметь их вовсе. Класс, у которого нет родителей, но есть потомки, называется базовым (base) или корневым (root), а тот, у которого нет потомков, - листовым (leaf). О классе, у которого есть только один родитель, говорят, что он использует одиночное наследование (Single inheritance); если родителей несколько, речь идет о множественном наследовании (Multiple inheritance)

Чтобы показать что один из классов реализует поведение, специфицированное в другом классе используют *реализацию*.

Для объединения часто повторяющихся групп блоков используют *пакеты* (Packages). В пакет можно поместить структурные, поведенческие и даже другие группирующие сущности. В отличие от компонентов, существующих во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только во время разработки.

С целью повышения информативности диаграммы можно использовать заметки (Note).

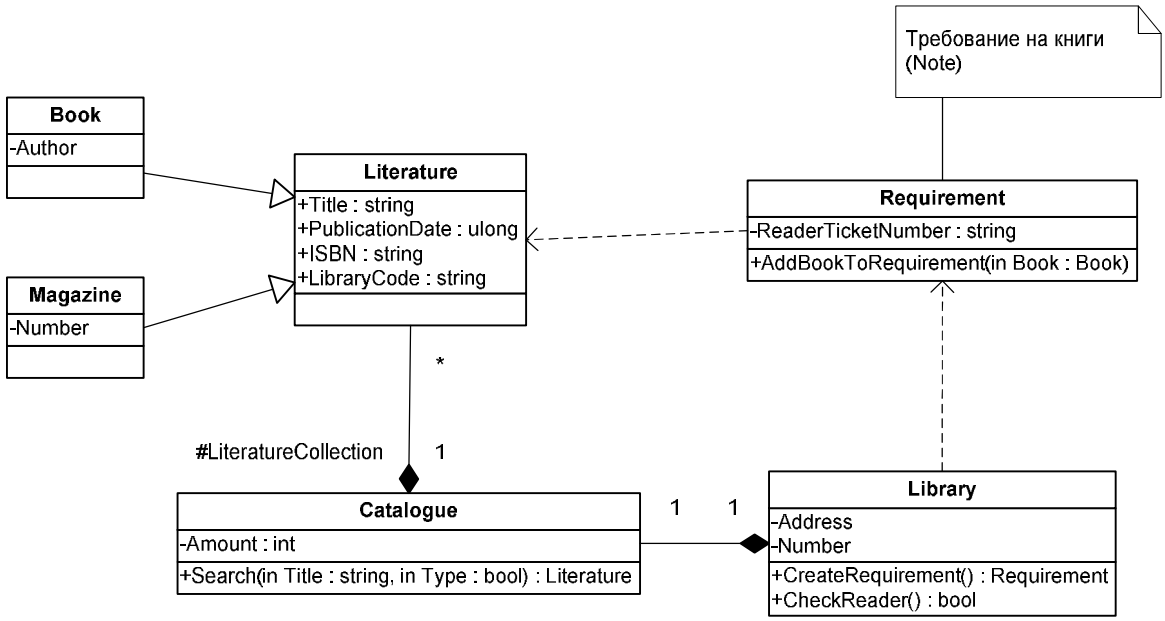

Рисунок 6. Фрагмент диаграммы классов ИС "Библиотека"

3. Порядок выполнения работы

Провести анализ предметной области на основе прецедентной модели, построенной в предыдущей лабораторной работе. Выделить основные сущности предметной области, их атрибуты и поведение, определить взаимосвязи между сущностями. Построить диаграмму классов анализируемой предметной области.

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.

3. Перечень функций системы полученный в результате анализа диаграммы прецедентов
4. Словесное описания сущностей системы, полученный в результате анализа диаграммы прецедентов (включая описание атрибутов классов и функций, реализующих его поведение).
5. Описание взаимоотношений между классами.
6. Диаграмма классов.
7. Выводы по работе.

5. Контрольные вопросы

1. Понятие и назначение диаграммы классов
2. Перечислить основные элементы, выносимые на диаграмму классов. Пояснить назначение каждого их них
3. Перечислить виды отношений между классами. Раскрыть суть каждого из них
4. Описать порядок выделения классов предметной области и их вынесение на диаграмму классов

6. Варианты индивидуальных заданий

Задания к лабораторной работе даны в методических указаниях к лабораторной работе №1 и являются ее продолжением в данной работе.

Лабораторная работа №3

ДИНАМИЧЕСКОЕ МОДЕЛИРОВАНИЕ РАБОТЫ СИСТЕМЫ

1. Цель работы

Изучить методы анализа поведения компонентов программной системы путем синтеза диаграмм, описывающих поведение моделируемой ИС с использованием нотации UML.

2. Общие сведения

Для большинства разрабатываемых систем, кроме самых простых и тривиальных, статических представлений совершенно недостаточно для моделирования процессов функционирования подобных систем как в целом, так и их отдельных подсистем и элементов.

Чтобы понять это, достаточно рассмотреть любой реальный объект окружающего мира. Понятно, что кроме сведений о его структуре отдельный интерес представляет и то, как и чем этот объект представлен во время своей «жизни». Это важно понимать и оговаривать, когда над ним возможно выполнять различные действия, в зависимости от того, в каком со-

стоянии он находится. Например, можно ввести такие значимые состояния для объекта автомобиль, как «движется» и «стоит». Очевидно, что операция «ремонтировать автомобиль» неприменима, когда он находится в состоянии «движется».

Объекты программных систем, являясь отражением реальных объектов предметной области, также нуждаются анализе и моделировании их поведения, а также взаимодействия между собой во времени. Нотация UML поддерживает в своем составе специальный набор диаграмм, которые обеспечивают динамическое представление работы элементов системы и их состояния. Фактически, для динамического представления системы в UML существует три вида диаграмм: диаграммы состояний (Statechart diagramm), диаграммы последовательности (Sequence diagramm) и диаграммы кооперaций (Collaboration diagramm).

Диаграммы состояний (Statechart diagram). Отражает внутренние состояния объекта в течение его жизненного цикла от момента создания объекта до его разрушения. Изображается в виде графа. Каждый объект рассматривается как сущность, которая контактирует с окружающим миром при помощи обмена сообщениями. Например, если читатель хочет взять в библиотеке книгу, он должен сначала запросить каталог (найти его), таким образом, он посылает запрос на нахождение книги. Состояния объекта связаны с некоторыми событиями. Определение состояния объекта зависит от самого объекта и от уровня моделирования. Диаграммы состояний используются для моделирования динамики поведения класса, поэтому ее целесообразно создавать только для объектов, имеющих несколько состояний. Этой диаграмме не ставится в соответствие программный код.

Состояние (State) описывает период времени в течение жизни объекта некоторого класса. Оно может быть описано тремя дополняющими друг друга способами: изменение значений переменных объекта; как время ожидания объектом некоторого события или времени, когда это событие произойдет; или же как период времени, в течение которого объект совершает некоторое продолжающееся действие.

Состояние может иметь имя, хотя довольно часто оно не имеет имени и описывается действиями, которые совершаются объектом в этом состоянии. Состояние может быть расширено: под названием состояния можно указать действия, выполняемые объектом в данном состоянии.

Событие (Event) – заслуживающий внимания случай, который произошел в данное время в данном месте. Оно не имеет длительности, то есть рассматривается как мгновенно произошедшее. Событие может иметь

некоторый набор параметров, которые характеризуют его и бывает исходящим и входящим.

Переход (Transition) объекта из одного состояния в другое отображается направленной стрелкой. Переход характеризуется входными событиями, ограничивающими условиями, аргументами, действиями и посылаемыми событиями. Ограничивающие условия показывают, при каких условиях совершается данный переход. Это необязательный элемент, но он часто используется особенно при автоматическом переходе, поскольку делает диаграмму более понятной.

История (History) – свойство объекта запоминать предыдущие состояния, которое отображается на диаграмме специальным значком, размещаемым внутри элемента «Состояние». Этот элемент не может использоваться без связи с состоянием, к которому относится история.

Начальное состояние – это состояние объекта, в котором он создается.

Конечное состояние – состояние, из которого объект не может вернуться в активное состояние.

Рассмотрим использование диаграмм состояний на примере состояний объекта «Заказ» в ИС «Интернет-магазин». Один из возможных бизнес-процессов работы ИС с заказом представлен ниже:

1. Пользователь (покупатель) оформил заказ, указав свои реквизиты, способ оплаты и доставки. Объект «Заказ» создан в системе (объект в памяти и/или запись в БД). Поскольку оплата заказа производится через внешнюю платежную систему, заказ не обрабатывается до момента получения подтверждения об оплате.
2. От платежной системы пришло сообщение об отказе в оплате или истек максимальный срок ожидания оплаты. Покупателю на указанный им e-mail отправляется соответствующее уведомление, после чего переходим к п. 6.
3. От платежной системы пришло подтверждение об оплате. Объект «Заказ» помечается как оплаченный. Адрес, указанный покупателем при оформлении заказа, передается во внешнюю систему проверки почтовых адресов; заказ ожидает подтверждения о проверке.
4. От системы проверки адресов пришло сообщение об ошибке в адресе. Покупателю на указанный им e-mail отправляется соответствующее уведомление; платежной системе отправляется сообщение о возврате платежа. После этого переходим к п. 6.
5. От системы проверки адресов пришло подтверждение корректности адреса. Заказ помечается как доставляемый и переходит в состояние ожи-

дания подтверждения о доставке. Покупателю отправляется уведомление о передаче заказа в службу доставки.

6. Заказ помечается как отмененный.

Пример диаграммы состояний для объекта «Заказ» представлен на рисунке 7.

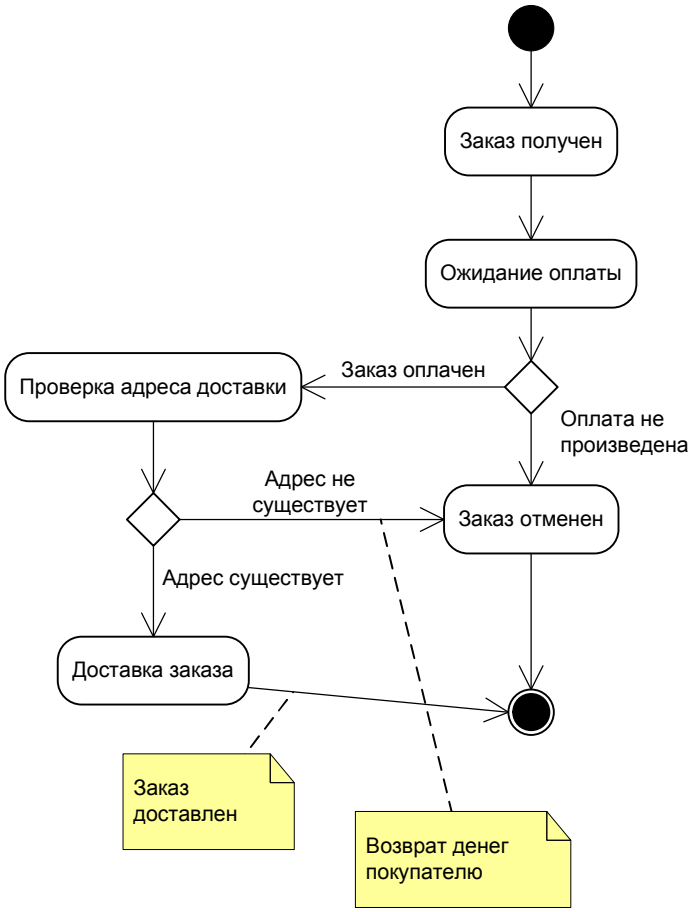

Рисунок 7. Диаграмма состояний для объекта «Заказ»

Диаграмма последовательностей (Sequence diagramm). Диаграмма последовательностей является диаграммой взаимодействия, отражающей поток событий, происходящих при реализации одного из вариантов использования. На этой диаграмме изображаются действующие лица, объекты, а также посылаемые и принимаемые ими сообщения. При построении в первую очередь принимается во внимание временная последовательность происходящих событий. На диаграмме последовательностей отображаются следующие элементы:

Линия жизни объекта (Lifeline) – отображает все, что происходит с объектом от его создания до его разрушения.

Объект (Object) – в рамках диаграммы последовательности – это экземпляр класса, сущность, шаблон.

Сообщение – поведение, связанное с передачей некоторой информации от одного объекта к другому, при котором передаваемая информация инициирует некоторое действие. Например, посылку сигнала, вызов некоторой операции, создание или уничтожение объекта. Сообщения бывают трех типов:

- *Сообщение (Message)* – горизонтальная стрелка от одного объекта к другому;
- *Сообщение с задержкой (Message with delivery time)* – наклонная стрелка от одного объекта к другому;
- *Сообщение объекта самому себе (Self message)* – направленная стрелка, начинающаяся и заканчивающаяся на одном и том же объекте.

Стимулирующее воздействие – взаимодействие между двумя объектами, которое передает информацию в предположении, что будет выполняться некоторое действие. Вызывает некоторую выполняемую операцию, а также создание или уничтожение объекта.

Активация – отображает период времени, в течение которого объект выполняет некоторое действие непосредственно или с помощью некоторой подчиненной процедуры. Это понятие характеризует продолжительность выполнения некоторого действия, которому в данный момент передано управление.

Время перехода используется для определения временных характеристик сообщения, которые могут использоваться в записи ограничений и условий. Функции времени могут включаться в логические выражения, истинность которых определяет последующий поток управления.

Пример диаграммы последовательностей для рассматриваемого нами примера «Заказ».

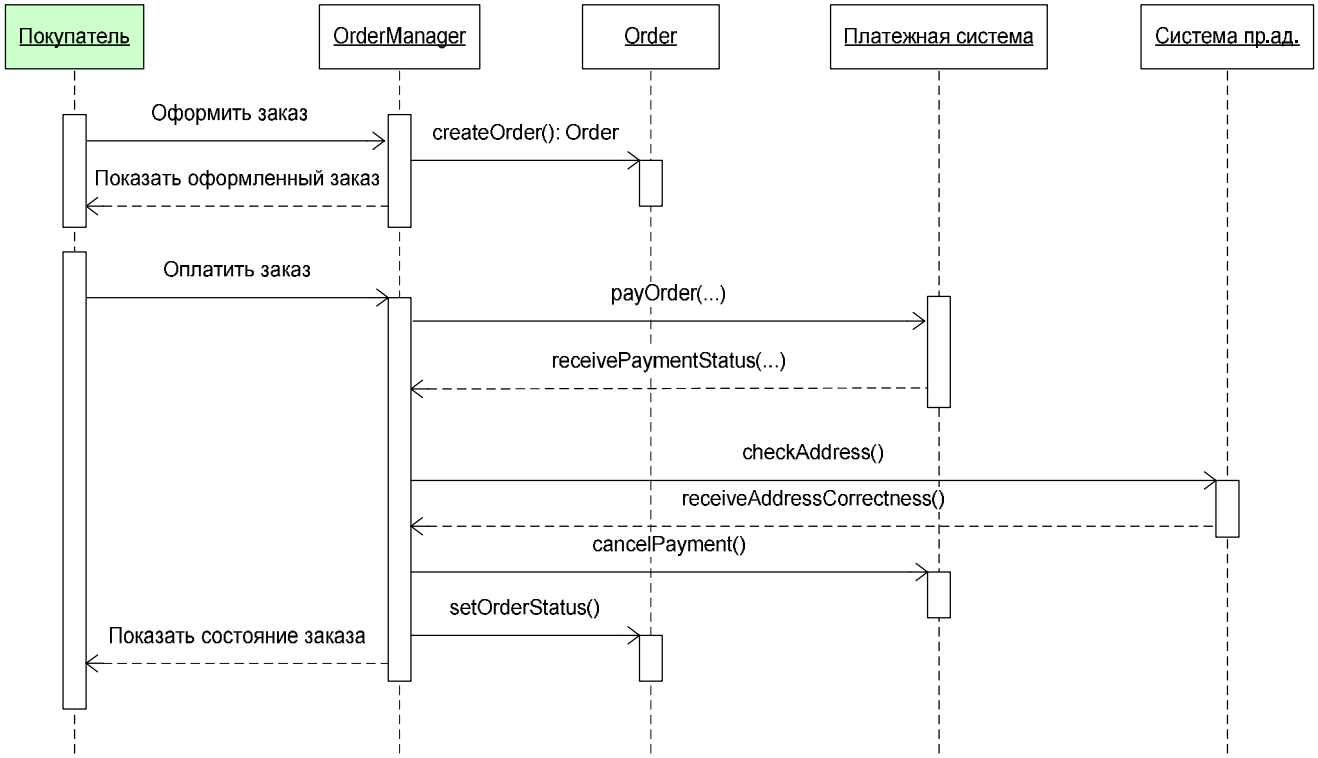

Рисунок 8. Диаграмма последовательностей

На приведенной выше диаграмме последовательностей **Покупатель** является *актером*, а не объектом системы. В некоторых вариантах нотации UML изображается в виде человечка. **OrderManager** – класс ИС, реализующий подсистему работы с заказами. **Order** – заказ. **Платежная система** и **Система пр.ад.** (проверки адресов) – внешние ИС по отношению к проектируемой; в проектируемой ИС фигурируют как внешние веб-сервисы.

3. Порядок выполнения работы

1. Произвести анализ предметной области. Выделить сущности предметной области, определить их взаимосвязи, атрибуты и поведение.
2. Разработать и построить диаграмму состояний в соответствии с вариантом индивидуального задания, используя диаграмму классов.
3. Разработать диаграмму последовательностей при помощи ранее разработанных диаграмм прецедентов.

4. Содержание отчета

1. Цель работы.
2. Вариант индивидуального задания.
3. Описания состояний, переходов (включая условия перехода).
4. Описание взаимодействий.

5. Диаграммы состояний и последовательностей.
6. Выводы по работе.

5. Контрольные вопросы

1. Пояснить суть динамического моделирования программной системы. Указать назначение и цель моделирования

2. Перечислить и охарактеризовать основные виды диаграмм, предназначенных для динамического моделирования системы

3. Что такое линия жизни объекта

6. Варианты индивидуальных заданий

Для выбранной ранее предметной области разработать диаграммы состояний, последовательности и кооперации.

ЗАКЛЮЧЕНИЕ

В ходе выполнения первой части работ данных методических рекомендаций у студента должно сформироваться четкое представление о современных подходах к проектированию и разработке программных систем с использованием нотации UML и соответствующих CASE средств. На примере представленных вариантов индивидуальных заданий это позволит изучить все особенности разноуровневого подхода к моделированию программной системы.

Вторая часть работ позволяет закрепить на практике приемы разработки и реализации Web-приложений с использованием языка C# и среды разработчика Microsoft Visual Studio. Знания, полученные студентами по итогам изучения теоретических разделов и применения их на практике в этой части могут быть использованы в последующих курсах при изучении дисциплин программирования и технологий проектирования и разработки программных систем.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Петцольд, Ч. Программирование в тональности C# / Ч. Петцольд: Русская редакция, 2004 – 512 с.: ил. – ISBN 5-7502-0180-5
2. Просиз, Дж. Программирование для Microsoft .NET / Дж. Просиз: – М., Русская редакция, 2003 – 680 с.: ил. – ISBN 5-7502-0217-8
3. Вендров, А. Проектирование программного обеспечения экономических информационных систем / А. М. Вендров: М., ФиС, 2005 – 352 с.: ил. – ISBN 5-279-02144-X;
4. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Гради Буч: Бином, 2001 – 560 с.: ил. – ISBN 5-7989-0067-3

ОГЛАВЛЕНИЕ