МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Государственное образовательное учреждение высшего профессионального образования
УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ПРОЕКТИРОВАНИЕ АРХИТЕКТУР ИНФОРМАЦИОННЫХ СИСТЕМ

Методические указания к лабораторным работам
по дисциплине «Архитектура информационных систем»
для студентов, обучающихся по специальности 230110165 «Вычислительные машины, комплексы, системы и сети»

Составитель К. С. Беляев

Ульяновск
2010

УДК 681.3 (076)
ББК 32.973я7
П 79

Рецензент кандидат технических наук, доцент кафедры «ИС» ФИСТ Ульяновского государственного технического университета О. Н. Евсеева.

Одобрено секцией методических пособий научно-методического совета университета

П 79

Проектирование архитектур информационных систем : методические указания к лабораторным работам/ сост. К. С. Беляев. – Ульяновск : УлГТУ, 2010. – 48 с.

Рассмотрены вопросы применения на практике в проектировании информационных систем подходов и методов, позволяющих получать их архитектуры. Методические указания построены по принципу выдачи заданий на лабораторные работы и приведения комментариев и примеров к их выполнению. Лабораторные работы взаимосвязаны между собой и предполагают последовательное выполнение.

Приведенные задания на лабораторные работы могут использоваться в рамках дисциплины «Архитектура информационных систем» для студентов, обучающихся по специальности 23010165 «Вычислительные машины, системы, комплексы и сети».

Сборник подготовлен на кафедре «Вычислительная техника».

УДК 681.3 (076)
ББК 32.973я7

Учебное издание
Проектирование архитектур информационных систем
Методические указания к лабораторным работам

Составитель БЕЛЯЕВ Константин Сергеевич
Редактор Н. А. Евдокимова

Подписано в печать 26.11.2010. Формат 60×84/16.
Усл. печ. л. 2.79. Тираж 100 экз. Заказ 1342.

Ульяновский государственный технический университет
432027, г. Ульяновск, ул. Сев. Венец, д. 32.

Типография УлГТУ, 432027, г. Ульяновск, ул. Сев. Венец, д. 32

Оглавление

Введение

Разработка информационной системы состоит из трех этапов: анализа, проектирования и реализации, в результате итеративного выполнения которых происходит пошаговое «наращивание» системы.

На этапах анализа и проектирования происходит построение архитектуры будущей информационной системы.

Архитектура программного обеспечения системы или набора систем состоит из всех важных проектных решений по поводу структур программы и взаимодействий между этими структурами, которые составляют системы. Проектные решения обеспечивают желаемый набор свойств, которые должна поддерживать система, чтобы быть успешной. Проектные решения предоставляют концептуальную основу для разработки системы, ее поддержки и обслуживания.

Цель данных методических указаний – изучение на практике применения в проектировании подходов и методов, позволяющих получать успешные архитектуры информационных систем. Методические указания построены по принципу выдачи заданий на лабораторные работы и приведения комментариев и примеров к их выполнению. Лабораторные работы взаимосвязаны между собой и предполагают последовательное выполнение.

Приведенные задания на лабораторные работы могут использоваться в рамках дисциплины «Архитектура информационных систем» для студентов, обучающихся по специальности 23010165 «Вычислительные машины, системы, комплексы и сети».

1. Лабораторная работа №1 «Установление требований»

Задание

Предложить для разработки информационную систему (ИС). ИС должна представлять собой программный комплекс, наделенный функциональностью, автоматизирующей конкретную деятельность в рамках предметной области, для которой разрабатывается система. Примером таких систем могут служить:

- автоматизированные системы управления (АСУ)
- электронные магазины, аукционы
- веб-порталы
- сервисы

Что надо сделать?

Составить документ описания требований к разрабатываемой ИС согласно шаблону (см. рис.1).

1.1 Установление требований

1.1.1 Документ описания требований

Документ, описывающий требования, является осязаемым результатом этапа установления требований. Большинство организаций вырабатывает документ описания требований в соответствии с заранее определенным шаблоном. Шаблон определяет структуру (содержание) и стиль документа.

Ядро документа описания требований состоит из формулировок (изложения) требований. Требования могут быть сгруппированы в виде *формулировок сервисов* (зачастую называемых функциональными требованиями) и *формулировок ограничений*. Формулировки сути сервисов могут быть затем разделены на *требования к функциям* (*function requirements*) и *требования к данным* (*data requirements*). (В литературе термин «*функциональные требования*» (*functional requirements*) в широком и в узком смысле используется как взаимозаменяемый. При использовании в узком смысле он соответствует тому, что мы называем требованиями к функциям).

Не говоря уже о самих требованиях, документ описания требований должен обращаться к проектным вопросам. Обычно проектные вопросы рассматриваются в начале документа, а затем в конце документа.

Во вводной части документа рассматривается бизнес-контекст проекта, включая цель проекта, участников проекта и основные ограничения. Ближе к заключительной части документа поднимаются другие проектные вопросы, включая план-график выполнения проектных работ, бюджет, риски, документацию и т. д.

1.1.2 Шаблоны документа

Шаблоны для документов описания требований широко доступны. Их можно найти в учебниках, стандартах, выпускаемых такими организациями как ISO, IEEE и т. д., на Web-страницах консалтинговых фирм, программных средствах разработки и т. д. Со временем каждая

организация разрабатывает свои собственные стандарты, которые соответствуют принятой в организации практике, корпоративной культуре, кругу читателей, типам разрабатываемых систем и т. д.

Шаблон документа описания требований определяет структуру документа и содержит подробные указания о содержании каждого из разделов документа. Указания могут включать содержание вопросов, мотивацию, примеры и дополнительные соображения.

На рис. 1 показано типичное оглавление документа описания требований. Последующие разделы включают объяснение к приведенному оглавлению.

1.1.3 Предварительные замечания к проекту

Часть документа описания требований, содержащая предварительные замечания к проекту, преимущественно дает ориентиры тем руководителям и участникам проекта, ответственным за принятие решений, которые, вероятно, не станут подробно изучать документ целиком. В начале документа необходимо ясно обозначить цели и рамки проекта, а затем описать деловой контекст системы.

Документ описания требований должен создать прецедент для системы. В частности, необходимо упомянуть обо всех усилиях, приложенных для обоснования необходимости системы на этапе планирования системы. Документ описания требований должен прояснить вопрос о том, каким образом предлагаемая система может способствовать достижению деловых целей и решению задач организацией.

Необходимо обозначить участников проекта системы. Важно, чтобы заказчик выступал не в виде безликого подразделения или офиса — необходимо привести конкретные имена. К концу дня человек должен быть в состоянии решить, приемлемо ли поставляемое программное обеспечение (ПО) для организации.

Документ описания требований

Содержание документа

1. Предварительные замечания к проекту

1.1. Цели и рамки проекта

1.2. Деловой контекст

1.3. Участники проекта

1.4. Идеи в отношении решений

1.5. Обзор документа

2. Системные сервисы

2.1. Рамки системы

2.2. Функциональные требования

2.3. Требования к данным

3. Системные ограничения

3.1. Требования к интерфейсу

3.2. Требования к производительности

3.3. Требования к безопасности

3.4. Эксплуатационные требования

3.5. Политические и юридические требования

3.6. Другие ограничения

4. Проектные вопросы

4.1. Открытые вопросы

4.2. Предварительный план-график

4.3. Предварительный бюджет

Приложения

Глоссарий

Деловые документы и формы

Ссылки

Рис.1 Содержание документа описаний требований

Хотя документ описания требований может быть как угодно далек от технических решений, все же важно обсудить идеи, касающиеся

решения на самых ранних этапах жизненного цикла (ЖЦ) разработки. Особый интерес представляют готовые решения. Всегда неплохо рассмотреть вариант приобретения готового продукта вместо его разработки «с нуля».

Документ описания требований должен предоставлять перечень существующих программных пакетов и компонент, которые должны быть в дальнейшем изучены в качестве вариантов возможных решений. Обратите внимание, что приобретение готового решения изменяет процесс разработки, однако это не избавляет от необходимости проведения анализа требований и проектирования системы!

Наконец, неплохо в заключение раздела предварительных замечаний к проекту документа описания требований привести обзор оставшейся части документа. Это может подтолкнуть к тому, чтобы изучить остальные части документа, а также способствует лучшему пониманию содержания документа. Обзор также может содержать пояснения в отношении методологии анализа проектирования, выбранной разработчиками.

1.1.4 Системные сервисы

Основная часть документа описания требований посвящена определению системных сервисов. Эта часть может занимать до половины всего объема документа. Это также, пожалуй, единственная часть документа, которая может содержать обобщенные модели — модели бизнес-требований.

Рамки системы можно моделировать с помощью диаграммы контекста. В пояснениях к диаграмме контекста должны быть четко определены рамки системы. Без подобного определения проект не может быть застрахован от попыток «растянуть» его рамки.

Функциональные требования можно моделировать с помощью диаграммы бизнес-прецедентов. Однако диаграмма охватывает перечень

функциональных требований только в самом общем виде. Все требования необходимо обозначить, классифицировать и определить.

Требования к данным можно моделировать с помощью диаграммы бизнес-классов. Так же, как и в случае функциональных требований, диаграмма бизнес-классов не дает полного определения структур данных для бизнес-процессов. Каждый бизнес-класс требует дальнейших пояснений. Необходимо описать атрибутное наполнение классов и определить идентифицирующие атрибуты классов. В противном случае невозможно правильно представить ассоциации.

1.1.5 Системные ограничения

Системные сервисы определяют, что должна делать система. Системные ограничения определяют, насколько система ограничена при выполнении обслуживания. Системные ограничения связаны со следующими видами требований.

- Требования к интерфейсу.
- Требования к производительности.
- Требования к безопасности.
- Эксплуатационные требования.
- Политические и юридические требования.

Требования к интерфейсу определяют, как система взаимодействует с пользователями. В документе описания требований определяется только «впечатление и ощущение» от GUI-интерфейса.

Начальное проектирование (закрашивание экрана) GUI-интерфейса проводится во время спецификации требований и позже во время системного проектирования.

В зависимости от области приложения требования к производительности могут играть довольно значительную роль в успехе проекта. В узком смысле они задают скорость (время отклика системы), с которой должны выполняться различные задания. В широком смысле, требования к производительности включают другие ограничения —

в отношении надежности, готовности, пропускной способности и т. д.

Требования к безопасности описывают пользовательские права доступа к информации, контролируемые системой. Пользователям может быть предоставлен ограниченный доступ к данным или ограниченные права на выполнение определенных операций с данными.

Эксплуатационные требования определяют программно-техническую среду, если она известна на этапе проектирования, в которой должна функционировать система. Эти требования могут оказывать влияние на другие стороны проекта, такие как подготовка пользователей и сопровождение системы.

Политические требования и юридические требования скорее подразумеваются, чем явно формулируются в документе описания требований. Подобная ошибка может обойтись очень дорого. Пока эти требования не выведены явно, программный продукт может быть трудно развернуть по политическим или юридическим причинам.

Возможны и другие виды ограничений. Например, в отношении некоторых систем могут предъявляться повышенные требования к легкости их использования (требования в отношении пригодности к использованию) или легкости их сопровождения (требования в отношении пригодности к сопровождению).

Значение выработки недвусмысленных определений для системных ограничений трудно переоценить. Существует немало примеров проектов, которые провалились из-за упущенных или неверно понятых ограничений. Эта проблема в равной мере относится как к заказчикам, так и к разработчикам. Недобросовестные или нерассудительные разработчики могут разыграть «карту системных ограничений», чтобы получить преимущество в своем стремлении уклониться от ответственности.

1.1.6 Проектные вопросы

Заключительная часть документа описания требований обращается к другим проектным вопросам. Один из важных разделов этой части называется «Открытые вопросы».

Здесь поднимаются все вопросы, которые могут сказаться на успехе проекта и которые не рассматривались в других разделах документа. Сюда относится ожидаемое возрастание значения некоторых требований, которые в текущий момент выходят за рамки проекта. Сюда можно отнести также любые потенциальные проблемы и отклонения в поведении системы, которые могут начаться в связи с развертыванием системы.

В этой же части необходимо представить предварительный план-график выполнения основных проектных заданий. Сюда же относится предварительное распределение людских и других ресурсов. Для выработки стандартных плановых графиков можно использовать программные средства управления проектами, например, такие как система PERT (program evaluation_and_review technique — метод оценки и пересмотра планов) или карты Ганта.

Прямым результатом составления план-графика может быть разработка предварительного бюджета. Стоимость проекта может быть выражена скорее в виде диапазона значений затрат, а не конкретного значения. При наличии надлежащим образом документированных требований для оценки затрат можно использовать один из подходящих методов.

1.1.7 Приложения

Приложения содержат остальную, полезную для понимания требований, информацию. Основным добавлением здесь служит глоссарий. Глоссарий определяет термины, сокращения и аббревиатуры, используемые в документе описания требований. Значение толкового глоссария трудно переоценить. Неверное истолкование терминологии таит в себе большую опасность для проекта.

Одна из особенностей, которую часто упускают из виду при составлении документа описания требований, состоит в том, что в проблемной области, определяемой документом, можно довольно неплохо разобраться с помощью изучения документов и форм, используемых в процессах делопроизводства. При возможности следует включать в документ заполненные формы — «пустые» формы не дают такого же уровня понимания бизнес-процессов.

Раздел ссылок содержит перечень документов, которые упоминаются или используются при подготовке документа описания требований. К ним могут относиться книги и другие опубликованные источники информации, но — что, пожалуй, даже более важно — необходимо также упомянуть протоколы совещаний, служебные записки и внутренние документы.

1.2 Пример документа описания требований

Документ описания требований ИС «Домашняя бухгалтерия»

1. Предварительные замечания к проекту

1.1. Цели и рамки проекта

Целью данного проекта является разработка информационной системы для ведения и оптимизации семейного бюджета. ИС «Домашняя бухгалтерия» должна быть проста в использовании и не требовать от пользователя знаний бухгалтерского учета.

1.2. Деловой контекст

Многие семьи в наше время планируют семейный бюджет. Ведение семейного бюджета при помощи подручных средств – карандаш, бумага – не всегда удобно и всегда трудоемко. Использование для этих целей компьютерных программ для ведения бухгалтерии не оправдано с точки зрения сложности их освоения и избыточного функционала для ведения домашней бухгалтерии. В связи с этим

возникает необходимость создания специализированной программы ведения домашней бухгалтерии.

1.3. Участники проекта

Заказчик – Васильева Марья Федоровна (m.vasileva@mypochta.ru)

Разработчик – Петров Степан Николаевич (petrov@coolsoft.com)

1.4. Идеи в отношении решений

Программа должна быть реализована в виде настольного приложения для операционных систем семейств MS Windows.

1.5. Обзор документа

В разделе «Системные сервисы» описывается, что должна делать система. В разделе «Системные ограничения» определяется, насколько система ограничена при выполнении обслуживания.

В разделе «Проектные вопросы» освещаются прочие проектные вопросы.

2. Системные сервисы

2.1. Рамки системы

Рамки системы можно моделировать с помощью диаграммы контекста.

Рис.2 Контекстная диаграмма ИС «Домашняя Бухгалтерия»

ИС «Домашняя Бухгалтерия» получает данные о доходах и расходах от внешней сущности «Домохозяин». Для передачи этих данных сущности «Домохозяин» должен авторизоваться. В своей работе сущность «Домашняя Бухгалтерия» использует информацию о ценах на товары и тарифах и курсах валют, получаемую от внешних сущностей «База тарифов и цен на товары» и «База курса валют». Результаты своей работы ИС «Домашняя Бухгалтерия» может отображать как внешней сущности «Домохозяин», так и генерировать в виде отчетов формата MS Excel для внешней сущности «MS Excel».

2.2. Функциональные требования

ИС должна обеспечивать следующие функциональные возможности:

- учет расходов;
- учет доходов;
- учет денег, отданных и взятых в долг;
- погашение долгов частями;
- проценты по долгам;
- контроль возврата долгов;
- система напоминания по долгам;
- составление бюджета расходов и доходов;
- планирование расходов;
- планирование доходов;
- система счетов;
- возможность использовать до пяти валют включительно;
- получение курсов валют из интернет;
- обмен валют;
- импорт данных из файлов Microsoft Excel;
- поиск по базе данных;
- фильтры и быстрый поиск по базе данных;

- экспорт данных в Excel, XML, текстовый файл;
- перенос данных;
- резервное копирование;
- печать данных;
- построение отчетов и диаграмм;
- настройка пользовательского интерфейса.

2.3. Требования к данным

ИС должна хранить свои данные в специализированных XML-файлах.

3. Системные ограничения

3.1. Требования к интерфейсу

ИС должна иметь стандартный интерфейс приложений, разработанных для ОС MS Windows.

3.2. Требования к производительности

Особых требований к производительности ИС нет.

3.3. Требования к безопасности

С программой могут работать несколько человек, входя в программу под своими именами. Для обеспечения конфиденциальности каждое имя можно защитить паролем. Добавление, изменение и удаление пользователей осуществляется в администраторе пользователей.

3.4. Эксплуатационные требования

ИС должна функционировать на ОС Windows XP, ОС Windows Vista, ОС Windows 7. Минимальные аппаратные требования определяются минимальными аппаратными требованиями к выше-перечисленным ОС.

3.5. Политические и юридические требования

Нет.

3.6. Другие ограничения

Нет.

4. Проектные вопросы

4.1. Открытые вопросы

Нет.

4.2. Предварительный план-график

1.09.2010 – 1.10.2010 – Анализ и установление требований к ИС

1.10.2010 – 1.11.2010 – Спецификация требований к ИС

1.11.2010 – 1.12.2010 – Кодирование ИС

1.12.2010 – 31.12.2010 – Тестовая эксплуатация ИС

11.01.2011 – 13.12.2011 – Ввод в эксплуатацию

4.3. Предварительный бюджет

Пятьдесят тысяч рублей.

5. Приложения

Глоссарий

ИС – информационная система

ОС – операционная система

Деловые документы и формы

Нет.

Ссылки

Нет.

1.2.1 Варианты заданий

1. АСУ деятельностью отдела кадров предприятия
2. АСУ складского хранения
3. АСУ деятельностью библиотеки
4. Веб-магазин по продаже часов
5. Веб-магазин по продаже фотоаппаратов
6. АСУ деятельностью аптечной сети
7. Веб-сайт букмекерской конторы

8. ИС учета успеваемости студентов
9. Веб-магазин по продаже компьютерных комплектующих
10. Программный RSS-агрегатор
11. Веб RSS-агрегатор
12. ИС «Ежедневник»
13. АСУ деятельностью магазина видеопроката
14. АСУ деятельностью автосалона
15. Веб-магазин по продаже одежды
16. ИС «Почтовый коллектор»
17. АСУ деятельностью магазина бензозаправки
18. АСУ учетом пациентов в поликлинике
19. АСУ учетом коммунальных платежей
20. АСУ деятельностью службы такси
21. ИС сбора и обработки ошибок (багтрекер)
22. Веб-сайт кафедры
23. Веб-сайт факультета
24. ИС хранения и каталогизации фотографий
25. ИС «Каталог недвижимости»

2. Лабораторная работа №2 «Спецификация требований»

Задание.

Составить спецификацию установленных в лабораторной работе №1 требований для проектируемой ИС. Для составления спецификации использовать язык UML. Модели спецификации разделить на три группы:

- Модели состояний
- Модели поведений
- Модели изменения состояний

Что надо сделать?

Построить UML-диаграммы к каждой группе моделей проектируемой системы и написать комментарии к ним. UML-диаграммы необходимые построить: вариантов использования системы, деятельности, взаимодействия объектов, классов и развертывания.

Примечание

Построение UML-диаграмм желательно осуществлять с помощью CASE-средств (StarUML, Rational Rose, MS Visio 2003 и др.).

2.1 Спецификация требований

Требования необходимо специфицировать (т. е. задать) графически или каким-либо иным формальным способом. Всесторонняя спецификация системы может потребовать использования многих типов моделей. Язык UML изобилует интегрированными методами моделирования, способными помочь бизнес-аналитику справиться с этой задачей. Спецификация — подобно процессу разработки ПО в целом — итеративный процесс с пошаговым наращиванием уровня детализации моделей. Немаловажную роль в успешном моделировании играет использование CASE-средств.

В результате спецификации требований вырабатываются три категории моделей: модели состояний, модели поведения и модели изменения состояния. Для каждой из категорий существует несколько методов работы с ними. Далее объясняются и иллюстрируются на примерах все основные методы моделирования языка UML.

Несмотря на то, что мы начнем изучение с моделей состояния, затем перейдем к моделям поведения, а затем — к моделям изменения состояний, это не отражает реальной последовательности, в которой проводится моделирование. Многие модели разрабатываются параллельно и служат источником взаимного развития. Это особенно справедливо в отношении двух основополагающих типов моделей — моделей классов и моделей прецедентов.

2.1.1 Принципы спецификации требований

Спецификация требований связана с доскональным моделированием требований заказчиков, определенных в процессе установления требований. При этом рассматриваются только услуги, которые стремятся получить от системы заказчики (формулировки сервисов). На этапе спецификации требований формулировки ограничений не подлежат

дальнейшей проработке, хотя и могут претерпеть изменения как результат обычного цикла итерации.

В качестве входной информации процесса спецификации требований выступают неформальные требования заказчиков, а результатом этого процесса являются модели спецификации проектных конструкций. Эти модели дают более формальное определение различных сторон (представлений) системы. Обычно требования пользователей в процессе спецификации подразделяются на две основные категории: функциональные требования и требования к данным.

В качестве результата этапа спецификации выступает расширенный («детально проработанный») документ описания требований. Новый документ часто называют *документом спецификации требований* (или просто «спецификацией» на жаргоне разработчиков). Структура исходного документа не изменяется, однако содержание значительно расширяется за счет глав, которые определяют требования заказчиков. Постепенно для целей проектирования и реализации документ спецификации требований заменяет документ описания требований (на практике расширенный документ может по-прежнему называться документом описания требований).

Модели спецификаций можно разделить на три группы.

1. Модели состояний.
2. Модели поведения.
3. Модели изменения состояний.

Модели состояний «детализируют» требования к данным. Модели поведения обеспечивают детализированные спецификации для функциональных требований. Модели изменения состояний охватывают два вида требований. Они призваны объяснить, каким образом действие функций приводит к изменению данных.

Модели представляются в виде диаграмм на языке визуального моделирования (Visual Modeling Language) — в нашем случае это язык UML. Обычно диаграмма служит целям моделирования одной из сторон

системы — состояний, поведения или изменения состояний. Заметное исключение составляет диаграмма классов, которая определяет все три аспекта — состояние и поведение объектов, и, косвенно, изменения состояний объектов.

Каждая диаграмма дает представление об определенной стороне системы. Взятые вместе диаграммы дают возможность разработчикам и пользователям взглянуть на предлагаемое решение с разных точек зрения, выделяя одни его стороны и игнорируя другие. Ни одна из диаграмм в отдельности не дает полного определения системы. Систему можно понять только через взаимосвязанный набор диаграмм.

Аналогично случаю интерпретации завершенных моделей конструирование диаграмм — это не последовательный процесс построения одной диаграммы за другой.

Диаграммы разрабатываются параллельно, и в результате каждой последующей итерации к ним добавляются новые детали. В то время, как разработчики должны следовать строго определенному процессу разработки, решение о том, какая из моделей должна играть роль «движущей силы» разработки, в значительной мере зависит от личных предпочтений аналитика. Обычно диаграммы прецедентов и модели классов — как наиболее важные типы моделей — конструируются параллельно, взаимно «обогащая» друг друга идеями.

С каждой новой итерацией разработки глубина и степень детализации спецификации возрастает. Многие более глубокие свойства объектов модели выражаются скорее в текстовом, нежели графическом виде. Некоторые свойства определяют замысел объекта модели, а не результат анализа. Некоторые другие свойства могут отражать особенности CASE-средств.

2.1.2 Спецификации состояний

Состояние объекта определяется значениями его атрибутов и ассоциаций. Например, объект BankAccount (Банковский счет) может

находиться в состоянии «превышение кредитного лимита», если значение атрибута balance (баланс) отрицательно. Поскольку состояния объекта определяются структурам данных, модели структур данных называются *спецификациями состояний*. Спецификация состояний дает статический взгляд на систему (поэтому моделирование состояний часто называют *статическим моделированием*). Здесь основной задачей является определение классов проблемной области, их атрибутов и отношений с другими классами. Вначале операции классов обычно не рассматриваются. Они выводятся из моделей спецификации поведения.

В типичной ситуации сначала определяются классы-сущности, т. е. классы, которые определяют проблемную область и характеризуются постоянным присутствием в базе данных системы. Подобные классы иногда называются «бизнес-классами». Классы, которые обслуживают системные события (управляющие классы) и классы, которые представляют GUI-интерфейс (классы представления или пограничные классы), не устанавливаются до тех пор, пока не станут известны поведенческие характеристики системы.

2.1.3 Информационная система «Запись на университетские курсы»

В качестве примера для спецификации требований выбрана информационная система «Запись на университетские курсы».

Постановка задачи.

Средний по размерам университет проводит набор студентов и аспирантов дневной и вечерней форм обучения для подготовки по ряду специальностей. Учебная структура университета состоит из факультетов. Факультет имеет в своем составе несколько кафедр. Хотя обучение и присвоение степени по определенной специальности является прерогативой факультета, обучение по специальности может включать дисциплины, преподаваемые на других факультетах.

В действительности, университет гордится свободой, предоставляемой им студентам в выборе дисциплин, изучаемых для получения специальности.

Гибкость выбора учебных курсов оказывает дополнительную нагрузку на университетскую систему набора. Индивидуально подобранным программам обучения должны быть противопоставлены правила, регулирующие получение степени по выбранной специальности. Такие правила можно сформулировать, например, в виде структуры дисциплин, изучение которых является обязательной предпосылкой получения диплома, так, чтобы студент мог прослушать обязательные для данной специальности курсы. Выбор студентами дисциплин может быть ограничен несоответствием расписаний, максимальной вместимостью аудиторий и т. д.

Гибкость в получении образования, предлагаемого университетом, стала одной из причин роста количества студентов. Однако для сохранения своих традиционно сильных сторон система набора – по-прежнему, частично ручная – должна быть заменена новым программным решением. Предварительно поиск готовых программных пакетов не дал результатов. Университетская система набора достаточно уникальна, чтобы оправдать разработку ПО собственными силами.

От системы требуется оказывать помощь в предварительной деятельности по записи на курсы и управляться с процедурами набора. Предварительная деятельность по записи должна включать рассылку оценок последнего семестра студентам наряду с инструкциями по записи на лекции. В период записи на курсы система должна принимать заявленные студентами программы обучения и проверять их на предмет обязательности, конфликтов расписания, вместимости аудиторий, специальных санкций и т. д. Решения по некоторым проблемам могут требовать консультаций с научными руководителями или профессорами, ответственными за преподавание дисциплин.

2.1.4 Выявление классов

Два разных аналитика, как правило, не могут прийти к идентичным моделям классов для одной и той же проблемной области, и точно так же два разных аналитика не пользуются одним и тем же мыслительным процессом при выделении классов. Литература изобилует подходами, предлагаемыми для выявления классов. Аналитики могут поначалу даже

следовать одному из этих подходов, однако последующие итерации, как правило, обязательно приводят к использованию нешаблонных и в чем-то даже случайных механизмов. Ниже перечислены эти подходы.

1. Подход на основе использования именных групп.
2. Подход на основе использования общих шаблонов для классов.
3. Подход на основе использования прецедентов.
4. Подход CRC (class–responsibility–collaborators – класс – обязанности – «сотрудники»).

2.1.5 Некоторые правила выявления классов

Ниже приведен далеко не полный перечень руководящих принципов или правил, которым должен следовать аналитик при выборе потенциальных классов. Вновь напоминаем о том, что здесь мы имеем дело только с классами-сущностями.

1. Для каждого класса должно быть ясно сформулировано его назначение в системе.

2. Каждый класс — это шаблон описания множества объектов. Единичные классы, для которых можно представить существование только одного объекта, весьма маловероятны среди «бизнес-объектов». Подобные классы обычно составляют в приложении «общее знание» и, как правило, жестко запрограммированы в программах приложения. Например, если система спроектирована для единственной организации, существование класса Organization (Организация) может быть не оправдано.

3. Каждый класс (т. е. класс-сущность) должен содержать набор атрибутов. Хорошим приемом является установление идентифицирующих атрибутов (ключей), чтобы помочь нам судить о мощности (cardinality) класса (т. е. ожидаемом количестве объектов данного класса в базе данных). Следует, однако, помнить о том, что класс не обязательно должен обладать пользовательским ключом.

4. Каждый класс должен отличаться от атрибута. Представляется ли понятие классом или атрибутом зависит от области приложения. Цвет автомобиля обычно воспринимается как атрибут класса Car (Автомобиль). Однако на фабрике по производству красок Color (Цвет) — это определенно класс со своими собственными атрибутами (яркостью, насыщенностью, прозрачностью и т. д.).

5. Каждый класс содержит набор операций. Однако на данном этапе мы не касаемся вопросов идентификации операций. Операции, входящие в интерфейс класса (сервисы, предоставляемые классом системе), являются логическим следствием формулировки назначения класса (пункт 1).

2.1.6 Пример выявления классов

Рассмотрим следующие требования к системе «Запись на университетские курсы» и выделим потенциальные классы.

1. *Для получения каждой университетской степени существует несколько обязательных и несколько выборочных курсов.*
2. *Каждому курсу соответствует заданный уровень и значение условных очков (CreditPoint – условное очко, начисляемое за прослушивание какого-либо курса (за один курс может быть начислено несколько очков), студент обязан на одном году набрать такое число курсов, чтобы число очков за них было не ниже определенного значения).*
3. *Курс может быть составной частью системы получения произвольного количества степеней.*
4. *Каждая степень определяет минимальное общее значение условных очков, требуемое для получения степени (например, для степени бакалавра (вычислительные и информационные системы) требуется 68 очков, включая обязательные курсы).*
5. *Студенты могут составлять из дисциплин программы обучения, приспособленные к их индивидуальным нуждам и обеспечивающие им получение степени, на которую они претендуют.*

Давайте проанализируем требования с целью выделения потенциальных классов. В первом утверждении подходящими классами являются классы Degree (Степень) и Course (Курс). Эти два класса удовлетворяют пяти правилам, перечисленным ранее.

Мы пока не уверены, должен ли и каким образом класс Course быть сужен до классов CompulsoryCourse (Обязательный курс) и ElectiveCourse (Выборочный курс). Например, ясно, что курс является обязательным или выборочным в зависимости от степени. Возможно, что различие между обязательными и выборочными курсами может быть зафиксировано с помощью ассоциации или даже атрибута класса. Таким образом, CompulsoryCourse и ElectiveCourse рассматриваются как нечеткие классы.

Второе утверждение идентифицирует только атрибуты класса Course, а именно course_level (уровень курса) и credit_point_value (количество условных очков). Третье утверждение характеризует ассоциацию между классами Course и Degree. Четвертая формулировка вводит атрибут min_total_credit_points (минимальное общее количество условных очков) в качестве атрибута класса Degree.

Последнее утверждение позволяет нам выделить три новых класса: Student (Студент), CourseOffering (Предлагаемый курс) и StudyProgram (Программа обучения). Первые два, безусловно, являются релевантными классами, а вот StudyProgram можно превратить в ассоциацию между классами Student и CourseOffering. Поэтому StudyProgram классифицируется как нечеткий класс. Наши рассуждения отражены в табл. 1.

Таблица 1

Релевантные классы	**Нечеткие классы**
Course (Курс)	ComplosoryCourse (Обязательный курс)
Degree (Степень)	ElectiveCourse (Выборочный курс)
Student (Студент)	StudyProgram (Программа обучения)
CourseOffering (Предлагаемый курс)	-

2.1.7 Спецификация классов

После того как перечень потенциальных классов сформирован, необходима их дальнейшая спецификация: классы требуется включить в диаграмму классов и определить их свойства. Некоторые свойства можно ввести и отобразить внутри графических пиктограмм, представляющих классы на диаграмме классов. Многие другие свойства, включенные в спецификацию класса, имеют только текстовое представление. CASE-средства, как правило, обладают возможностями редактирования, позволяющими легко вводить или модифицировать подобную информацию посредством диалоговых окон, снабженных вкладками, или с помощью аналогичных способов.

2.1.8 Выявление и спецификация атрибутов классов

Графическая пиктограмма, представляющая класс, состоит из трех отделений (имя класса, атрибуты, операции). Спецификация атрибутов классов принадлежит к спецификации состояний и рассматривается в этом разделе. Спецификация операций рассматривается позже.

Выделение атрибутов осуществляется параллельно с выделением классов. Идентификация атрибутов – своего рода «побочный эффект» установления классов. Это не означает, что выявление атрибутов — простая задача. Напротив, это процесс, требующий значительных усилий и многократных итераций.

Исходные модели спецификации определяют только атрибуты, являющиеся существенными для понимания состояний, в которых могут находиться объекты класса. Остальные атрибуты можно до поры до времени игнорировать (однако аналитик должен быть уверен в том, что установленная, но проигнорированная на определенном этапе информация не будет по ошибке утеряна и будет зафиксирована впоследствии). Маловероятно, чтобы все атрибуты класса были

приведены в документе описания требований, однако важно не включать в спецификацию те атрибуты, которые не вытекают из требований.

В последующих итерациях можно добавить больше атрибутов.

Для имен атрибутов мы рекомендуем придерживаться простого соглашения: в именах атрибутов использовать только строчные буквы, а слова в составных именах отделять подчеркиванием.

2.1.9 Пример спецификации классов

Снова обратимся к примеру системы «Запись на университетские курсы». Рассмотрим следующие дополнительные требования, изложенные в документе описания требований.

1. *Выбор студентам учебных курсов может быть ограничен из-за конфликтов расписания, а также за счет ограничения на количество студентов, которое может быть набрано на текущий предлагаемый курс.*
2. *Предлагаемая студентом программа обучения вводится в интерактивную систему записи на курсы. Система проверяет программу на непротиворечивость и сообщает о любых проблемах. Проблемы требуется решать при помощи научного руководителя. Окончательная программа является предметом научного согласования со стороны представителя заведующего кафедрой, а затем направляется секретарю учебного заведения.*

В первом утверждении упоминаются конфликты расписания, однако мы не знаем достоверно, как следует моделировать эту проблему. Возможно, что речь здесь идет о прецеденте, который процедурно определяет конфликты расписания. Вторую часть этой же формулировки можно смоделировать за счет ведения атрибута enrolment_quota (квота набора) в класс CourseOffering. Теперь также ясно, что класс должен обладать атрибутами year (год) и semester (семестр).

Вторая формулировка укрепляет нас во мнении о необходимости введения класса StudyProgram. Можно видеть, что класс StudyProgram сочетает в себе ряд дисциплин, предлагаемых к изучению в текущий момент. Поэтому класс StudyProgram также должен обладать атрибутами year и semester.

Ближайшее рассмотрение нечетких классов CompulsoryCourse и ElectiveCourse приводит нас к выводу, что учебный курс является обязательным или выборочным относительно определенной ученой степени. Один и тот же курс может быть обязательным по отношению к одной степени, выборочным, что касается другой, и вообще не допустимым применительно к некоторым другим степеням. Раз так, то CompulsoryCourse и ElectiveCourse не являются классами в полном смысле слова.

На рис.3 представлена модель классов, соответствующая проведенным нами рассуждениям. Кроме того, на рисунке используются символы (стереотипы (stereotype)) <<PK>> и <<CK>> для обозначения первичных ключей и потенциальных ключей, соответственно. Это уникальные идентификаторы объектов для рассмотренных классов. Здесь же заданы типы данных для атрибутов.

Классы StudyProgram и CourseOffering не имеют пока идентифицирующих атрибутов. Они будут введены в эти классы после установления ассоциативных связей между классами.

2.1.10 Выявление ассоциаций

Нахождение основных ассоциаций представляет собой побочный эффект процесса выявления классов. При определении классов аналитик принимает решение об атрибутах классов, и некоторые из этих атрибутов являются ассоциациями с другими классами. Атрибуты могут относиться к элементарным типам данных либо могут вводиться в качестве других классов, устанавливая таким образом отношения с другими классами.

По существу, любой атрибут, относящийся к неэлементарным типам данных, должен моделироваться как ассоциация (или агрегация) по классу, представляющему этот тип данных.

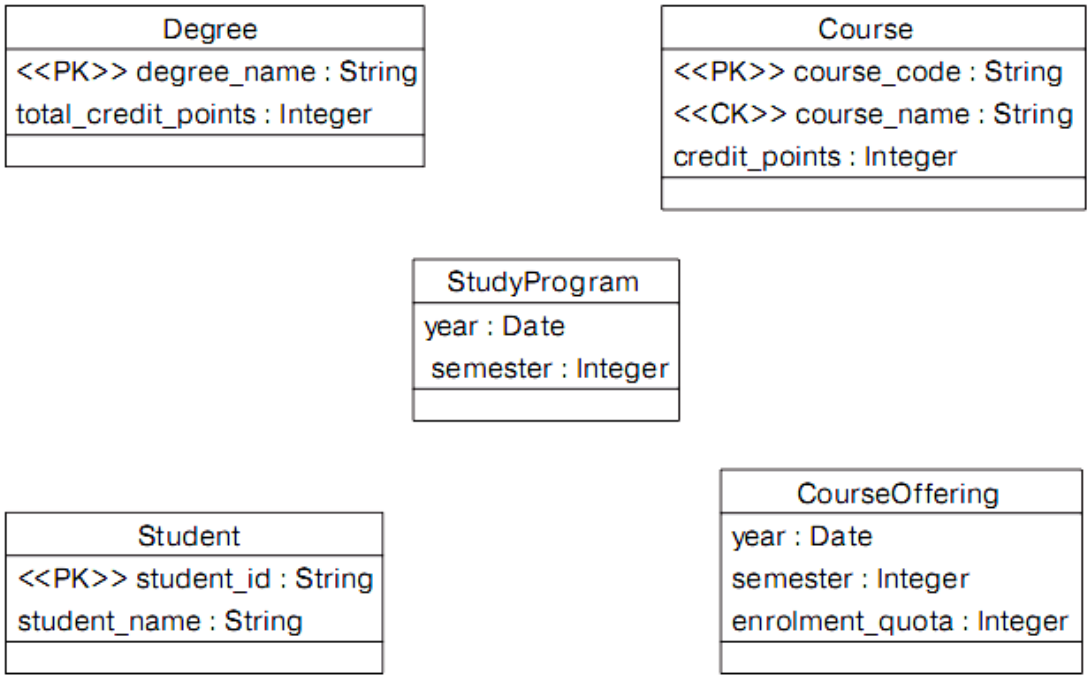

Рис.3 Модель классов

Выполнение пробного прогона прецедентов позволяет выявить остающиеся ассоциации. Устанавливаются пути взаимодействия между классами, необходимые для прогона прецедентов. Обычно ассоциации должны поддерживать эти пути взаимодействия.

Каждая тернарная ассоциация должна быть заменена циклом или бинарной ассоциацией. Тернарные ассоциации привносят риск неверного семантического истолкования.

Иногда для того чтобы полностью выразить базовую семантику, циклы, образуемые ассоциациями, не должны коммутировать (быть замкнутыми). Это значит, что по меньшей мере одна из ассоциаций в

цикле может быть производной (derived). Подобная ассоциация является избыточной в семантическом смысле и должна быть исключена (хорошая семантическая модель должна быть лишена избыточности). Вполне допустимо, что многие производные ассоциации все же войдут в проектную модель (например, из соображений эффективности).

2.1.11 Спецификация ассоциаций

Спецификация ассоциаций подразумевает выполнение следующих действий.

1. Присваивание имен ассоциациям.
2. Присваивание имен ассоциативным ролям.
3. Установление кратности ассоциации.

Правила именования ассоциаций должны соответствовать соглашениям по именованию атрибутов — имена ассоциаций состоят из строчных букв, отдельные слова в имени ассоциации разделяются подчеркиванием.

Если два класса связаны только одним ассоциативным отношением, задавать имя ассоциации и ассоциативные ролевые имена между этими классами необязательно. CASE-средства могут внутренне различать каждую ассоциацию через системные идентификационные имена.

Ролевые имена можно использовать для раскрытия более сложных ассоциаций, в частности самоассоциативных отношений (self associations) (рекурсивных ассоциаций, которые связывают объекты одного и того же класса). При задании ролевых имен их следует выбирать с учетом того, что в проектной модели они станут атрибутами классов, расположенных на противоположных концах ассоциативной связи.

2.1.12 Выявление агрегаций и композиций

Поиск агрегаций ведется параллельно с поиском ассоциаций. Если ассоциация проявляет одно или более из четырех семантических свойств, рассмотренных выше, то ее можно моделировать как агрегацию.

При объяснении отношения агрегации лакмусовой бумажкой выступают фразы «включает» («has») и «является частью» («is_part_of»). При истолковании отношения сверху-вниз по иерархии классов используется фраза «включает» (например, Книга «включает» Главу). При интерпретации снизу-вверх используется фраза «является частью» (например, Глава «является частью» Книги). Если предложение, описывающее отношение, прочитывается вслух с использованием этих фраз и оно лишено смысла на естественном языке, то это отношение не является агрегацией. Со структурной точки зрения агрегация часто связывает воедино большое количество классов, тогда как ассоциация степени выше двух бессмысленна. Когда требуется связать более двух классов воедино, отличным вариантом моделирования может быть агрегация типа Участник.

2.1.13 Спецификация агрегаций и композиций

Язык UML обеспечивает только ограниченную поддержку агрегации. Сильная форма агрегации называется в UML *композицией*.

В композиции составной объект может физически содержать компонентные объекты (семантически это отношение берется «по значению»). Компонентный объект может принадлежать только одному составному объекту. Отношение композиции языка UML в большей или меньшей степени соответствует нашим агрегациям типа Безраздельно обладает и Обладает.

Слабая форма агрегации в UML называется просто *агрегацией*. Это отношение семантически берется «по ссылке» — составной объект физически не содержит компонентный объект. Один компонентный

объект может обладать несколькими ассоциативными или агрегативными связями в модели. Попросту говоря, агрегация в языке UML соответствует нашим агрегациям типа Включает и Участник.

Сплошной ромб в языке UML представляет композицию. Пустой ромб используется для определения агрегации. В остальном спецификация агрегации совпадает с обозначениями для ассоциации.

2.1.14 Пример спецификации агрегации и композиции

Снова обратимся к системе Записи на университетские курсы. Рассмотрим следующие дополнительные требования.

1. *Академическая характеристика студента должна быть доступна по требованию. Характеристика должна включать информацию об оценках, полученных студентом по каждому из курсов, на которые записался студент (и которые он не покинул без взыскания, т. е. первые три недели с начала семестра).*
2. *За каждый курс отвечает преподаватель, однако этот курс могут вести и другие преподаватели. В течение разных семестров за один курс могут отвечать разные преподаватели; кроме того, в течение разных семестров курс могут вести разные преподаватели.*

На рис. 4 показана модель классов, в которой выделено отношение агрегации.

Понятие «студент» (объект Student) «включает» академическую характеристику (объект AcademicRecord) — это описание композиции в языке UML (с семантикой «по значению»). Каждый объект AcademicRecord физически помещен в один из объектов Student. Несмотря на существование ассоциации takes (брать [курс обучения]) объект AcademicRecord включает атрибут course_code (код курса).

Это необходимо, поскольку ассоциация takes реализуется с помощью атрибута takes_crsoff («берет» дисциплину) объекта Student, введенного как коллекция (collection), например, Set[CourseOffering].

Атрибут takes_crsoff не зависит от информации во вложенном объекте AcademicRecord, хотя он безусловно связывает объект Student с объектом Course.

Понятие «курс» (объект Course) включает дисциплину (объект CourseOffering)» — это описание агрегации в языке UML (с семантикой «по ссылке»). Каждый объект CourseOffering только логически содержится в одном из объектов Course. Объект CourseOffering может также участвовать в других агрегациях и/или ассоциациях (например, с объектами Student и AcademicInCharge (ответственный преподаватель)).

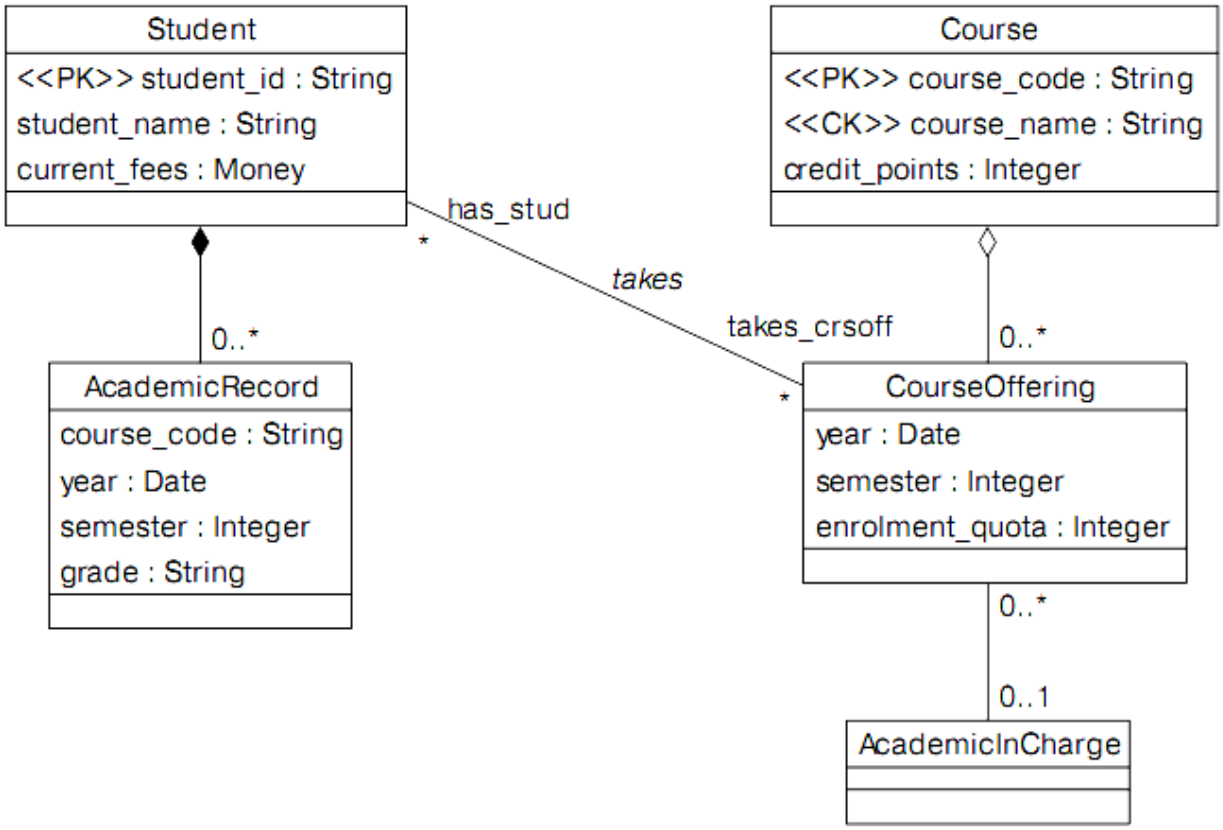

Рис.4 Модель классов с отношениями агрегации и композиции

2.1.15 Выявление обобщений

Многие суперклассы/подклассы аналитик отмечает еще в процессе формирования первоначального перечня классов. Многие другие обобщения можно обнаружить при определении ассоциаций.

Различные ассоциации (даже принадлежащие одному и тому же классу) может потребоваться связать с классом на различных уровнях обобщения/специализации. Например, класс Course может быть связан с классом Student (Student takes Course — студент берет курс), кроме того, этот класс может быть связан с классом TeachingAssistant (TeachingAssistant teaches Course — ассистент ведет курс). Дальнейший анализ может показать, что класс TeachingAssistant является подклассом Student.

При поиске отношения обобщения лакмусовой бумажкой выступают фразы «может быть» («can_be») и «это нечто вроде» («is_a_kind_of»). При истолковании отношения сверху-вниз по иерархии классов используется фраза «может быть» (например, Student «can_be» a TeachingAssistant — «студент «может быть» ассистентом»). При интерпретации отношения снизу-вверх используется фраза «это нечто вроде» (например, TeachingAssistant «is_a_kind_of» Student — «ассистент «это нечто вроде» студента»). Обратите внимание, что если также справедливо утверждение о том, что «ассистент — это «TeachingAssistant «is_a_kind_of» Teacher», то мы установили множественное наследование.

2.1.16 Спецификация обобщений

Отношение обобщения между классами показывает, что один класс совместно использует структуру или поведение, определенные в одном или более классов. Обобщение представляется в языке UML сплошной линией со стреловидным наконечником, указывающим на суперкласс.

Полная спецификация обобщения включает несколько мощных возможностей.

Например, более подробное определение отношения может содержать квалификацию доступа к нему, указания на то, предоставляет ли класс права другому классу, решая, что необходимо делать в случае множественного наследования и т. д.

2.1.17 Спецификация поведения

Поведение системы — так как оно выглядит для внешнего пользователя — изображается в вид прецедентов. Модели прецедентов можно разрабатывать на различных уровнях абстракции. Их можно применить к системе в целом для того, чтобы специфицировать основные функциональные блоки разрабатываемого приложения. Их также можно использовать для фиксации поведения пакетов UML, частей пакетов или даже класса внутри пакета.

На этапе анализа прецеденты вбирают в себя системные требования, концентрируясь на том, что делает или должна делать система. На этапе проектирования представление проектных решений в виде прецедентов можно использовать для спецификации поведения системы в том виде, как оно должно быть реализовано.

Поведение системы, закрепленное с помощью прецедентов, требует осуществить соответствующие вычисления и обеспечить взаимодействие объектов для выполнения этих прецедентов. Вычисления можно смоделировать с помощью диаграмм видов деятельности. Взаимодействие объектов можно задать с помощью диаграмм последовательностей или диаграмм кооперации.

Спецификация поведения позволяет взглянуть на систему с точки зрения ее функционирования. Здесь основная задача состоит в том, чтобы определить прецеденты для области приложений и установить, какие классы участвуют в выполнении этих прецедентов. При этом необходимо идентифицировать операции классов и сообщения, передаваемые между объектами. Хотя взаимодействие объектов инициирует изменения состояния объектов, спецификации поведения дают функциональный взгляд на застывшее состояние системы. Изменения в состоянии объектов явным образом находят выражение в спецификациях изменения состояний.

Модели прецедентов должны нарабатываться итеративно и параллельно с моделями классов. Классы, введенные в спецификации состояний, подлежат дальнейшей детальной проработке, при этом обозначаются наиболее важные операции. Следует, однако, напомнить, что спецификация состояний определяет только классы сущностей («бизнес-объектов»).

По мере создания моделей поведения появляются еще два уровня классов.

1. Классы, которые обслуживают события, инициируемые пользователями, и представляют бизнес-процессы *(управляющие классы)*.

2. Классы, представляющие GUI-интерфейсы *(пограничные классы)*.

2.1.18 Выявление прецедентов

При выявлении прецедентов аналитик должен убедиться в том, что он твердо придерживается сущности концепции прецедентов. Прецеденты представляют следующие компоненты общей модели системы.

1. *Завершенный* фрагмент функциональных возможностей (включая *основной поток* логики управления, его любые вариации (*подпотоки*) и исключительные условия (*альтернативные потоки*)).

2. Фрагмент внешне *наблюдаемых функций* (отличных от внутренних функций).

3. *Ортогональный* фрагмент функциональных возможностей (прецеденты могут при выполнении совместно использовать объекты, но выполнение каждого прецедента независимо от других прецедентов).

4. Фрагмент функциональных возможностей, *инициируемый субъектом* (будучи инициирован, прецедент может взаимодействовать с другими субъектами).

5. При этом возможно, что субъект окажется только на принимающем конце прецедента (может быть, опосредовано), инициированного другим субъектом. Фрагмент функциональных возможностей, который предоставляет *субъекту* ощутимый *полезный результат* (и этот полезный результат достигается в пределах одного прецедента).

Выявление прецедентов основано на анализе следующих источников информации.

1. Требования, определенные в документе описания требований.

2. Субъекты и их цели применительно к системе.

Напомним, что требования представляются в отпечатанном виде. При поиске прецедентов нас интересуют только функциональные требования.

Прецеденты можно определить на основе анализа задач, выполняемых субъектами.

Ответы на эти вопросы могут позволить обозначить прецеденты.

- Каковы основные задачи, выполняемые каждым субъектом?
- Должен ли субъект получать доступ к информации в системе или модифицировать информацию?
- Должен ли субъект информировать систему о любых изменениях в других системах?
- Должен ли субъект быть проинформирован о непредвиденных изменениях в системе?

В ходе анализа прецеденты обращаются к личностным потребностям субъектов. В некотором роде это прецеденты субъектов. Поскольку прецеденты определяют основные строительные блоки для системы, то существует необходимость в выделении системных прецедентов. Системные прецеденты извлекают все то общее, что присутствует в прецедентах субъектов, и дают возможность разработать общее решение, применимое (через механизм наследования) к области прецедентов субъектов. «Субъектом» системного прецедента является разработчик/программист, а не пользователь. Системные прецеденты идентифицируются на этапе проектирования.

2.1.19 Спецификация прецедентов

Спецификация прецедентов включает графическое представление субъектов, прецедентов и четырех типов отношений, перечисленных ниже.

1. Ассоциация.
2. Включение.
3. Расширение.
4. Обобщение прецедента.

Отношение ассоциации устанавливает каналы связи между субъектом и прецедентом. В качестве стереотипов для отношений «включает» (include) и «расширяет» (extend) используются слова <<include>> и <<extend>>. Отношение обобщения позволяет специализировать прецедент посредством изменения любого из аспектов базового прецедента.

Отношение включения <<include>> позволяет вынести общее поведение за пределы включаемого прецедента. (Это отношение пришло на смену широко применяемой в более ранних версиях UML концепции отношения <<uses>> (использует)). Отношение <<extend>> обеспечивает

контролируемую форму расширения поведения прецедента с помощью активизации другого прецедента в определенных точках расширения. Отношение <<include>> отличается от отношения <<extend>> в том, что «включаемый» прецедент является необходимым для завершения «активизирующего» прецедента.

На практике проект может легко столкнуться с проблемами, если прилагать слишком большие усилия для выявления отношений между прецедентами и установления, какие отношения применимы к определенной паре прецедентов. Кроме того, обобщенные прецеденты имеют тенденцию к таким тесным связям, что взаимосвязи станут превалировать и загромождать диаграмму, смещая акценты с надлежащей идентификации прецедентов в сторону отношений между прецедентами.

2.1.20 Пример спецификации прецедентов

Обращаясь к постановке задачи для системы «Запись на университетские курсы» на стр.23-24, а также к требованиям, определенным на стр. 26-27 и стр. 29-30 установим прецеденты на основе анализа функциональных требований.

На рис. 5 показана обобщенная диаграмма прецедентов для приложения «Запись на университетские курсы». Модель содержит четыре субъекта и четыре прецедента. Каждый прецедент инициируется субъектом и является завершенным, внешне видимым и ортогональным фрагментом функциональных возможностей. Все субъекты, за исключением субъекта Student, представляют собой инициирующих субъектов. Субъект Student получает результаты экзаменов и инструкции по записи на учебные курсы перед тем, как программа обучения в следующем семестре (учебном периоде) может быть введена и проверена.

Прецедент Provide Examination Results (Предоставить результаты экзаменов) может «расширить» (<<extend>>) прецедент Provide Enrolment Instructions (Предоставить инструкции по записи). Первый прецедент не

всегда расширяет последний прецедент. Например, для новых студентов результаты экзаменов неизвестны. Вот почему отношение моделируется с использованием стереотипа расширения (<<extend>>), а не включения (<<include>>).

Отношение <<include>> было установлено от прецедента Enter Program of Study (Ввести программу обучения) к прецеденту Validate Program of Study (Проверить программу обучения). Отношение <<include>> означает, что первый из прецедентов всегда включает последний. Как только программа изучения введена, она проверяется на предмет конфликтов расписания, специальных согласований и т.д.

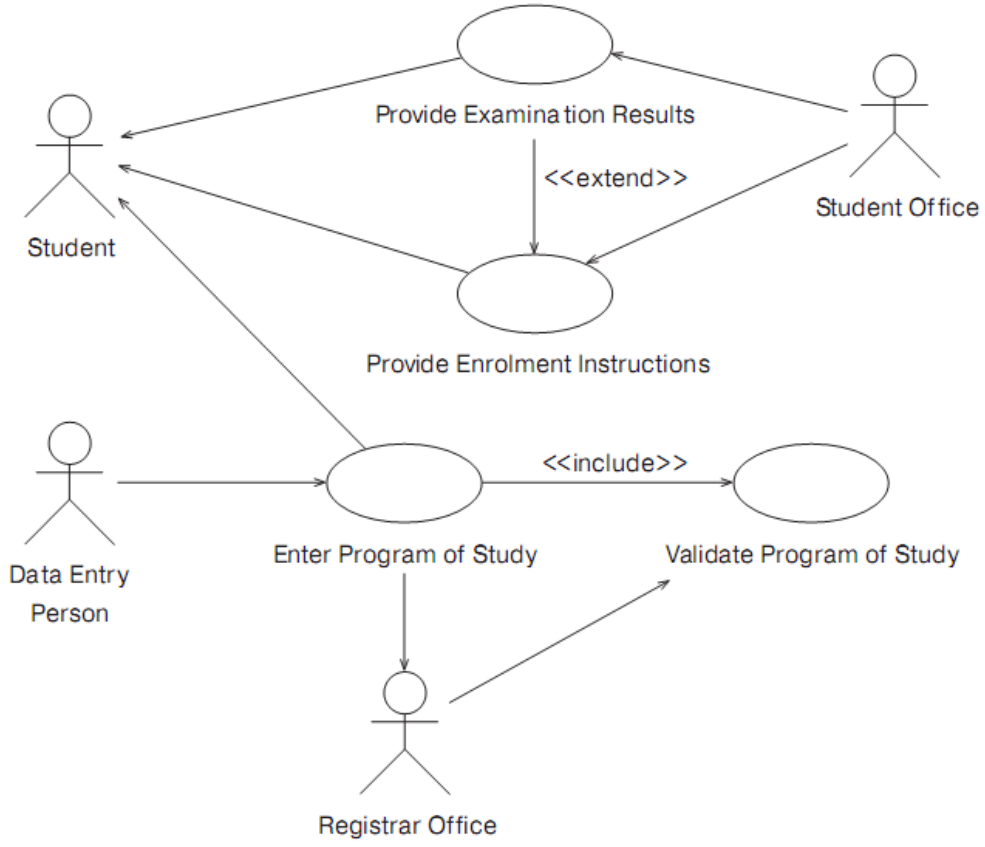

Рис.5 Обобщенная диаграмма прецедентов для приложения «Запись на университетские курсы»

2.1.21 Выявление видов деятельности

Каждый прецедент можно моделировать с помощью одного или нескольких графов видов деятельности. Событие, источником которого служит субъект, инициирующий прецедент, это то же самое событие, которое запускает выполнение графа видов деятельности. Процесс выполнения последовательно переходит от одного состояния вида деятельности к другому. Состояние вида деятельности считается завершенным, когда завершается его вычисление. Внешние инициируемые событиями прерывания, которые могут вызвать завершение состояния вида деятельности, допускаются только в исключительных случаях. Если ожидается, что подобные события могут происходить часто, то следует вместо этого воспользоваться диаграммой состояний.

Виды деятельности лучше всего выявлять на основе анализа предложений неформальной спецификации прецедентов. Каждая фраза, содержащая глагол, может рассматриваться как потенциальный вид деятельности. Описание альтернативных потоков вводит в граф видов деятельности ветвление и разделение потоков.

Они приводят к исключительным (непредвиденным) состояниям деятельности. Возможны также параллельные потоки управления.

2.1.22 Спецификация видов деятельности

После выявления состояний видов деятельности спецификация видов деятельности выглядит как довольно простой процесс соединения этих состояний линиями переходов.

Параллельные потоки инициируются (разделяются) и сливаются, что отображается на диаграмме в виде жирной полосы, обозначающей синхронизацию потоков. Альтернативные потоки создаются (разветвляются) и объединяются, что отображается в виде ромбов ветвления.

Внешние события на графе видов деятельности обычно отсутствуют. Однако существует графический метод включения внешних событий в граф. Аналогично существуют графические обозначения для состояний потоков объектов для представления объектов, которые являются входными или выходными для вида деятельности.

2.1.23 Выявление последовательностей сообщений

Выявление последовательностей сообщений является логическим продолжением моделирования видов деятельности. Виды деятельности, представленные на соответствующей диаграмме, отображаются на сообщения диаграммы последовательностей.

Если уровни абстракции, используемые для построения модели видов деятельности и модели последовательностей, совпадают, то осуществить отображение видов деятельности на сообщения довольно просто.

2.1.24 Спецификация последовательностей сообщений

При спецификации сообщений полезно проводить различие между сообщением, представляющим собой сигнал, и сообщением, которое являет собой вызов операции. Сигнал означает асинхронное взаимодействие объектов. Отправитель может продолжить выполнение потока сразу после отправки сигнального сообщения. Вызов означает синхронное обращение к операции с условием возврата управления отправителю. Возвращаемое сообщение может возвращать вызывающему объекту некоторые значения либо просто уведомлять его об успешном завершении операции. В последнем случае возвращаемое сообщение отображать на диаграмме последовательностей не обязательно.

2.1.25 Пример спецификация последовательностей

Приняв во внимание требования, изложенные на стр. 34-35, а также обращаясь к примеру спецификации прецедентов на стр. 41-42, построим диаграмму последовательностей для прецедента «Enter Program of Study».

На рис. 6 показана диаграмма последовательностей для приведенного выше сценария. Субъект Data Entry Person (Лицо, вводящее данные) инициирует прецедент, отправляя сообщение с запросом объекту граничного класса ProgramEntryWindow (Окно программы ввода) для того, чтобы добавить студента (обозначенного аргументом std) к списку записавшихся на курс (аргумент crs) в данном семестре (аргумент sem).

Объект класса ProgramEntryWindow проверяет с помощью объекта aStudent, имеет ли право студент записаться (например, внес ли студент s_check соответствующую плату). Выходной аргумент возвращает значение «yes» или «no» объекту: ProgramEntryWindow. Если возвращается значение «no», запись не может продолжаться, и объект :ProgramEntryWindow отправляет автосообщение самому себе, чтобы уничтожить (destroy) себя (операция деструктора). Запись условия в квадратных скобках говорит о том, что операция деструктора является необязательной.

Использование автосообщения для указания на необходимость уничтожения объекта является просто сокращенной записью.

В действительности объект_экземпляр не может уничтожить себя. Сообщение об уничтожении (destroy) должно быть отправлено объекту_классу. Кроме того, в языке UML имеется специальное обозначение для операции уничтожения объекта — большой знак X, помещенный на жизненной линии объекта в точке, в которой объект должен быть уничтожен.

Если студент aStudent может быть записан, нам требуется выяснить, открыта ли запись на курс aCourse. Для обслуживания этого требования

объект aCourse должен запросить свой текущий объект aCourseOffering (на который указывает агрегативная связь от Course к CourseOffering (рис. 6)). Если в текущем объекте aCourseOffering свободных мест уже нет, запись не может продолжаться, и объект :ProgramEntryWindow снова отправляет себе автосообщение, чтобы разрушить себя.

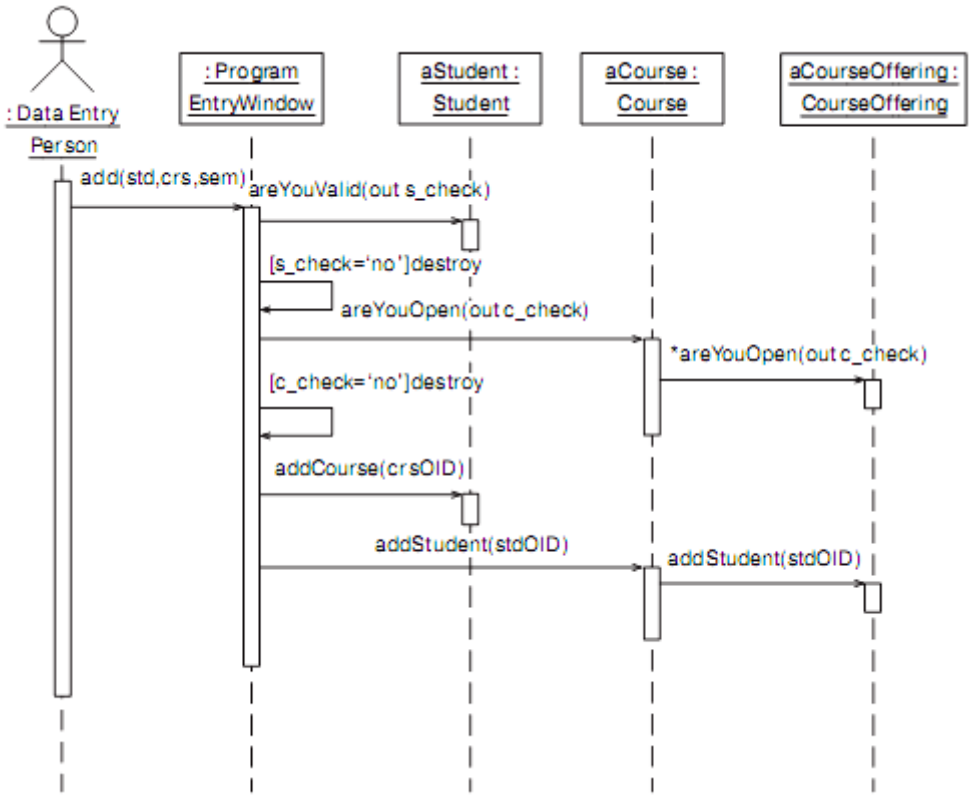

Рис.6 Диаграмма последовательностей

Если процесс записи может продолжаться, объект :ProgramEntryWindow требует от объекта aStudent добавить себе объект aCourseOffering, а затем требует, чтобы объект aCourseOffering добавил объект aStudent к себе. Последовательность двух операций объекта :ProgramEntryWindow может быть изменена на обратную, если программа гарантирует ссылочную целостность между aCourseOffering и aStudent (т. е. если студент записался на предлагаемый курс, то в список студентов по данному курсу должен быть внесен данный студент).

Заметим, что если для хранения объектов Student и CourseOffering используется ПО СУБД, то объект :ProgramEntryWindow мог бы отправлять только одно из этих двух сообщений, т. е. отправить либо сообщение addCourse, либо сообщение addStudent.

После того как объект aStudent добавлен к объекту aCourseOffering, ПО СУБД берет на себя ответственность за поддержание ссылочной целостности и должно зафиксировать существование связи от aStudent к aCourseOffering, и наоборот.

Заметим также, что фактические аргументы, включенные в сообщения addCourse и addStudent, представляют собой идентификаторы объектов (OID), содержащие значения OID (дескрипторы), указывающие на объект aStudent и aCourseOfferin, соответственно. Эти значения OID были определены, когда объект :ProgramEntryWindow получил доступ к объектам aStudent и aCourse с помощью сообщений areYouValid и areYouOpen, представляющих собой запросы на возможность записи и наличие мест на курсе.

Заключение

Выполнив лабораторные работы, данные в методических указаниях, студент на практике получит опыт применения в проектировании подходов и методов, которые позволят получать успешные архитектуры информационных систем. Справедливости ради, стоит заметить, что изложенные подходы и методы неединственные. Самостоятельно осваивая другие методики и применяя их совместно с ранее изученными, студент сможет построить еще более гибкие и эффективные архитектуры.

Библиографический список

1. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML.: Пер.с англ. – М. : Издательский дом «Вильямс»,2002. – 432 с.
2. Инженерия программного обеспечения, –6-е изд. –.: Пер. с англ. – М. : Издательский дом «Вильямс»,2002. – 624 с.
3. Принципы работы с требованиями к программному обеспечению. Унифицированный подход.: Пер. с англ. – М. : Издательский дом «Вильямс», 2002. – 448 с.
4. UML. Основы: Пер. с англ. – СПб. : Символ-Плюс, 2002. – 192 с.
5. ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы. – М. , 1990. URL: http://www.nist.ru/hr/doc/gost/34-602-89.htm
6. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. URL: http://ieeexplore.ieee.org/iel4/5841/15571/00720574.pdf