

Глава 1. Основные сведения о языке UML

Самое лучшее средство - это большая диаграмма, приколотая к стене.

Даг Скотт

1.1. Цели и история создания языка UML

Унифицированный язык моделирования UML (Unified Modeling Language) - это преемник того поколения методов объектно-ориентированного анализа и проектирования, которые появились в конце 80-х и начале 90-х годов. Создание UML фактически началось в конце 1994 г., когда Гради Буч и Джеймс Рамбо начали работу по объединению их методов Booch [Буч-1999] и OMT (Object Modeling Technique) под эгидой компании Rational Software. К концу 1995 г. они создали первую спецификацию объединенного метода, названного ими Unified Method, версия 0.8. Тогда же в 1995 г. к ним присоединился создатель метода OOSE (Object-Oriented Software Engineering) Ивар Якобсон. Таким образом, UML является прямым объединением и унификацией методов Буча, Рамбо и Якобсона, однако дополняет их новыми возможностями.

UML находится в процессе стандартизации, проводимом консорциумом OMG (Object Management Group), в настоящее время он принят в качестве стандартного языка моделирования и получил широкую поддержку. UML принят на вооружение практически всеми крупнейшими компаниями - производителями программного обеспечения (Microsoft, IBM, Hewlett-Packard, Oracle, Sybase и др.). Кроме того, практически все мировые производители CASE-средств, помимо Rational Software (Rational Rose), поддерживают UML в своих продуктах (Paradigm Plus (CA), System Architect (Popkin Software), Microsoft Visual Modeler и др.). Полное описание UML можно найти на сайтах <http://www.omg.org> и <http://www.rational.com>. Первое описание UML на русском языке содержится в книге [Фаулер-1999], в дальнейшем изложении терминология языка соответствует данному переводу. Кроме него, имеются также переводы [Боггс-2000], [Буч-2000] и [Ларман-2001].

1.2. Средства UML

Создатели UML представляют его как язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем и других систем различной природы. UML содержит стандартный набор диаграмм и нотаций самых разнообразных видов. Стандарт UML версии 1.1, принятый OMG в 1997 г., предлагает следующий набор диаграмм для моделирования:

- **диаграммы вариантов использования (use case diagrams)** -для моделирования бизнес-процессов организации и требований к создаваемой системе);
- **диаграммы классов (class diagrams)** - для моделирования статической структуры классов системы и связей между ними;
- **диаграммы поведения системы (behavior diagrams):**
 - **диаграммы взаимодействия (interaction diagrams):**
 - ◆ **диаграммы последовательности (sequence diagrams)** и
 - ◆ **кооперативные диаграммы (collaboration diagrams)** -для моделирования процесса обмена сообщениями между объектами;
 - **диаграммы состояний (statechart diagrams)** -для моделирования поведения объектов системы при переходе из одного состояния в другое;
 - **диаграммы деятельности (activity diagrams)** -для моделирования поведения системы в рамках различных вариантов использования, или моделирования деятельности;
- **диаграммы реализации (implementation diagrams):**
 - **диаграммы компонентов (component diagrams)** -для моделирования иерархии компонентов (подсистем) системы;
 - **диаграммы размещения (deployment diagrams)** -для моделирования физической архитектуры системы.

1.3. Диаграммы вариантов использования

Понятие варианта использования (use case) впервые ввел Ивар Якобсон и придал ему такую значимость, что в настоящее время вариант использования превратился в основной элемент разработки и планирования проекта.

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать.

Действующее лицо (actor) - это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима некоторая информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования.

Действующие лица делятся на три основных типа - пользователи системы, другие системы, взаимодействующие с данной, и время. Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

Для наглядного представления вариантов использования в качестве основных элементов процесса разработки программного обеспечения (ПО) применяются диаграммы вариантов использования. На рис. 1.1 показан пример такой диаграммы для банкомата (Automated Teller Machine, ATM).

На данной диаграмме человеческие фигурки обозначают действующих лиц, овалы - варианты использования, а линии и стрелки - различные связи между действующими лицами и вариантами использования.

На этой диаграмме показаны два действующих лица: клиент и кредитная система. Существует также шесть основных действий, выполняемых моделируемой системой: перевести деньги, сделать вклад, снять деньги со счета, показать баланс, изменить идентификационный код и осуществить оплату.

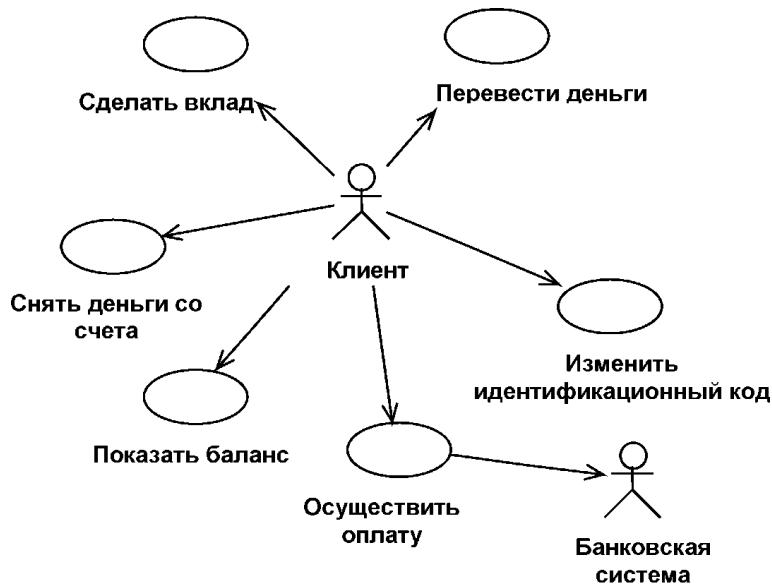


Рис. 1.1. Пример диаграммы вариантов использования

На диаграмме вариантов использования показано взаимодействие между вариантами использования и действующими лицами. Она отражает требования к системе с точки зрения пользователя. Таким образом, варианты использования - это функции, выполняемые системой, а действующие лица - это заинтересованные лица (stakeholders) по отношению к создаваемой системе. Такие диаграммы показывают, какие действующие лица инициируют варианты использования. Из

них также видно, когда действующее лицо получает информацию от варианта использования. Данная диаграмма, например, отражает взаимодействие между вариантами использования и действующими лицами системы ATM. В сущности, диаграмма вариантов использования иллюстрирует требования к системе. В нашем примере, клиент банка инициирует большое количество различных вариантов использования: «Снять деньги со счета», «Перевести деньги», «Сделать вклад», «Показать баланс» и «Изменить идентификационный код». От варианта использования «Осуществить оплату» стрелка указывает на Банковскую систему. Действующими лицами могут быть внешние системы, и потому в данном случае Банковская система показана именно как действующее лицо - она внешняя для системы ATM. Направленная от варианта использования к действующему лицу стрелка показывает, что вариант использования предоставляет некоторую информацию, используемую действующим лицом. В данном случае вариант использования «Осуществить оплату» предоставляет Банковской системе информацию об оплате по кредитной карточке.

Все варианты использования, так или иначе, связаны с внешними требованиями к функциональности системы. Варианты использования всегда следует анализировать вместе с действующими лицами системы, определяя при этом реальные задачи пользователей и рассматривая альтернативные способы решения этих задач.

Действующие лица могут играть различные роли по отношению к варианту использования. Они могут пользоваться его результатами или могут сами непосредственно в нем участвовать. Значимость различных ролей действующего лица зависит от того, каким образом используются его связи.

Конкретная цель диаграмм вариантов использования -это документирование вариантов использования (всё, входящее в сферу применения системы), действующих лиц (всё вне этой сферы) и связей между ними. Разрабатывая диаграммы вариантов использования, старайтесь придерживаться следующих правил:

- Не моделируйте связи между действующими лицами. По определению действующие лица находятся вне сферы действия системы. Это означает, что связи между ними также не относятся к её компетенции.
- Не соединяйте сплошной стрелкой (коммуникационной связью) два варианта использования непосредственно. Диаграммы данного типа описывают только, какие варианты использования доступны системе, а не порядок их выполнения. Для отображения порядка выполнения вариантов использования применяют диаграммы деятельности.
- Вариант использования должен быть инициирован действующим лицом. Это означает, что должна быть сплошная стрелка, начинающаяся на действующем лице и заканчивающаяся на варианте использования.

Хорошим источником для идентификации вариантов использования служат внешние события. Следует начать с перечисления всех событий, происходящих во внешнем мире, на которые система должна каким-то образом реагировать. Какое-либо конкретное событие может повлечь за собой реакцию системы, не требующую вмешательства пользователей, или, наоборот, вызвать пользовательскую реакцию. Идентификация событий, на которые необходимо реагировать, помогает идентифицировать варианты использования.

Варианты использования начинают описывать, что должна будет делать система. Чтобы фактически разработать систему, однако, потребуются более конкретные детали. Эти детали описываются в документе, называемом «поток событий» (flow of events). Целью потока событий является документирование процесса обработки данных, реализуемого в рамках варианта использования. Этот документ подробно описывает, что будут делать пользователи системы, и что - сама система.

Хотя поток событий и описывается подробно, он также не должен зависеть от реализации. Цель - описать, что будет делать система, а не как она будет делать это. Обычно поток событий включает:

- краткое описание;
- предусловия (pre-conditions);
- основной поток событий;
- альтернативный поток событий (или несколько альтернативных потоков);

- постусловия (post-conditions). Последовательно рассмотрим эти составные части. **Описание**

Каждый вариант использования должен иметь связанное с ним короткое описание того, что он будет делать. Например, вариант использования «Перевести деньги» системы ATM может содержать следующее описание:

Вариант Использования «Перевести деньги» позволяет клиенту или служащему банка переводить деньги с одного счета до востребования или сберегательного счета на другой.

Предусловия

Предусловия варианта использования - это такие условия, которые должны быть выполнены, прежде чем вариант использования начнет выполняться сам. Например, таким условием может быть выполнение другого варианта использования или наличие у пользователя прав доступа, требуемых для запуска этого. Не у всех вариантов использования бывают предварительные условия.

Ранее упоминалось, что диаграммы вариантов использования не должны отражать порядок их выполнения. С помощью предусловий, однако, можно документировать и такую информацию. Например, предусловием одного варианта использования может быть то, что в это время должен выполняться другой.

Основной и альтернативный потоки событий

Конкретные детали вариантов использования описываются в основном и альтернативных потоках событий. Поток событий поэтапно описывает, что должно происходить во время выполнения заложенной в варианты использования функциональности. Поток событий уделяет внимание тому, что будет делать система, а не как она будет делать это, причем описывает все это с точки зрения пользователя. Основной и альтернативный потоки событий включают следующее описание:

- способ запуска варианта использования;
- различные пути выполнения варианта использования;
- нормальный, или основной, поток событий варианта использования;
- отклонения от основного потока событий (так называемые альтернативные потоки);
- потоки ошибок;
- способ завершения варианта использования.

Например, поток событий варианта использования «Снять деньги» может выглядеть следующим образом:

Основной поток

1 . Вариант использования начинается, когда клиент вставляет свою карточку в ATM.
2. ATM выводит приветствие и предлагает клиенту ввести свой персональный идентификационный номер.

3. Клиент вводит номер.
4. ATM подтверждает введённый номер. Если номер не подтвержден, выполняется альтернативный поток событий A1 .

5. ATM выводит список доступных действий:
- положить деньги на счет;
 - снять деньги со счета;
 - перевести деньги.

6. Клиент выбирает пункт «Снять деньги».
7. ATM запрашивает, сколько денег надо снять.
8. Клиент вводит требуемую сумму.
9. ATM определяет, имеется ли на счету достаточно денег. Если денег недостаточно, выполняется альтернативный поток A2. Если во время подтверждения суммы возникают ошибки, выполняется поток ошибок E1 .

10. ATM вычитает требуемую сумму из счета клиента.
11. ATM выдает клиенту требуемую сумму наличными.
12. ATM возвращает клиенту его карточку.
13. ATM печатает чек для клиента.

1.4. Вариант использования завершается.

Альтернативный поток A1. Ввод неправильного идентификационного номера.

1. ATM информирует клиента, что идентификационный номер введен неправильно.

2. ATM возвращает клиенту его карточку.

3. Вариант использования завершается.

Альтернативный вариант использования A2. Недостаточно денег на счету.

1. ATM информирует клиента, что денег на его счету недостаточно.

2. ATM возвращает клиенту его карточку.

3. Вариант использования завершается.

Поток ошибок E1. Ошибка в подтверждении запрашиваемой суммы.

1. ATM сообщает пользователю, что при подтверждении запрашиваемой суммы произошла ошибка и дает ему номер телефона службы поддержки клиентов банка.

2. ATM заносит сведения об ошибке в журнал ошибок. Каждая запись содержит дату и время ошибки, имя клиента, номер его счета и код ошибки.

3. ATM возвращает клиенту его карточку.

4. Вариант использования завершается.

Постусловия

Постусловиями называются такие условия, которые всегда должны быть выполнены после завершения варианта использования. Например, в конце варианта использования можно пометить флажком какой-нибудь переключатель. Информация такого типа входит в состав постусловий. Как и для предусловий, с помощью постусловий можно вводить информацию о порядке выполнения вариантов использования системы. Если, например, после одного из вариантов использования должен всегда выполняться другой, это можно описать как постусловие. Такие условия имеются не у каждого варианта использования.

Связи между вариантами использования и действующими лицами

В языке UML на диаграммах вариантов использования поддерживается несколько типов связей между элементами диаграммы. Это связи коммуникации (communication), включения (include), расширения (extend) и обобщения (generalization).

Связь коммуникации - это связь между вариантом использования и действующим лицом. На языке UML связи коммуникации показывают с помощью односторонней ассоциации (сплошной линии со стрелкой). Направление стрелки позволяет понять, кто инициирует коммуникацию.

Связь включения применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования. С помощью таких связей обычно моделируют многократно используемую функциональность. В примере ATM варианты использования «Снять деньги» и «Положить деньги на счет» должны опознать (автентифицировать) клиента и его идентификационный номер перед тем, как допустить осуществление самой транзакции. Вместо того чтобы подробно описывать процесс аутентификации для каждого из них, можно поместить эту функциональность в свой собственный вариант использования под названием «Автентифицировать клиента».

Связь расширения применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования только при необходимости использовать функциональные возможности другого.

На языке UML связи включения и расширения показывают в виде зависимостей с соответствующими стереотипами, как показано на рис. 1.2.

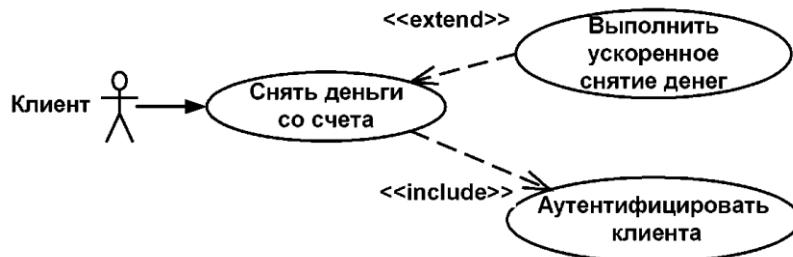


Рис. 1.2. Связи использования и расширения

С помощью связи обобщения показывают, что у нескольких действующих лиц имеются общие черты. Например, клиенты могут быть двух типов: корпоративные и индивидуальные. Эту связь можно моделировать с помощью нотации, показанной на рис. 1.3.

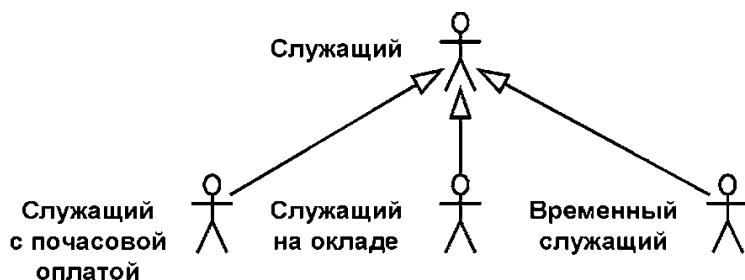


Рис. 1.3. Обобщение действующего лица

Нет необходимости всегда создавать связи этого типа. В общем случае, они нужны, если поведение действующего лица одного типа отличается от поведения другого поскольку это затрагивает систему. Если оба подтипа используют одни и те же варианты использования, показывать обобщение действующего лица не требуется.

Варианты использования являются необходимым средством на стадии формирования требований к ПО. Каждый вариант использования - это потенциальное требование к системе, и пока оно не выявлено, невозможно запланировать его реализацию.

1.4. Диаграммы взаимодействия

Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов.

Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Сообщение (message) - это средство, с помощью которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций.

Информационное (informative) сообщение - это сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) - это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

Императивное (imperative) сообщение - это сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует два вида диаграмм взаимодействия: диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams).

1.4.1. Диаграммы последовательности

Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования. Например, вариант использования «Снять деньги» предусматривает несколько

возможных последовательностей, такие как снятие денег, попытка снять деньги, не имея их достаточного количества на счету, попытка снять деньги по неправильному идентификационному номеру и некоторые другие. Нормальный сценарий снятия денег со счета (при отсутствии таких проблем, как неправильный идентификационный номер или недостаток денег на счете) показан на рис. 1.4.

Эта диаграмма последовательности показывает поток событий в рамках варианта использования «Снять деньги». Все действующие лица показаны в верхней части диаграммы; в приведенном выше примере изображено действующее лицо Клиент. Объекты, требуемые системе для выполнения варианта использования «Снять деньги», также представлены в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

На диаграмме последовательности объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия. Эта линия называется линией жизни (lifeline) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

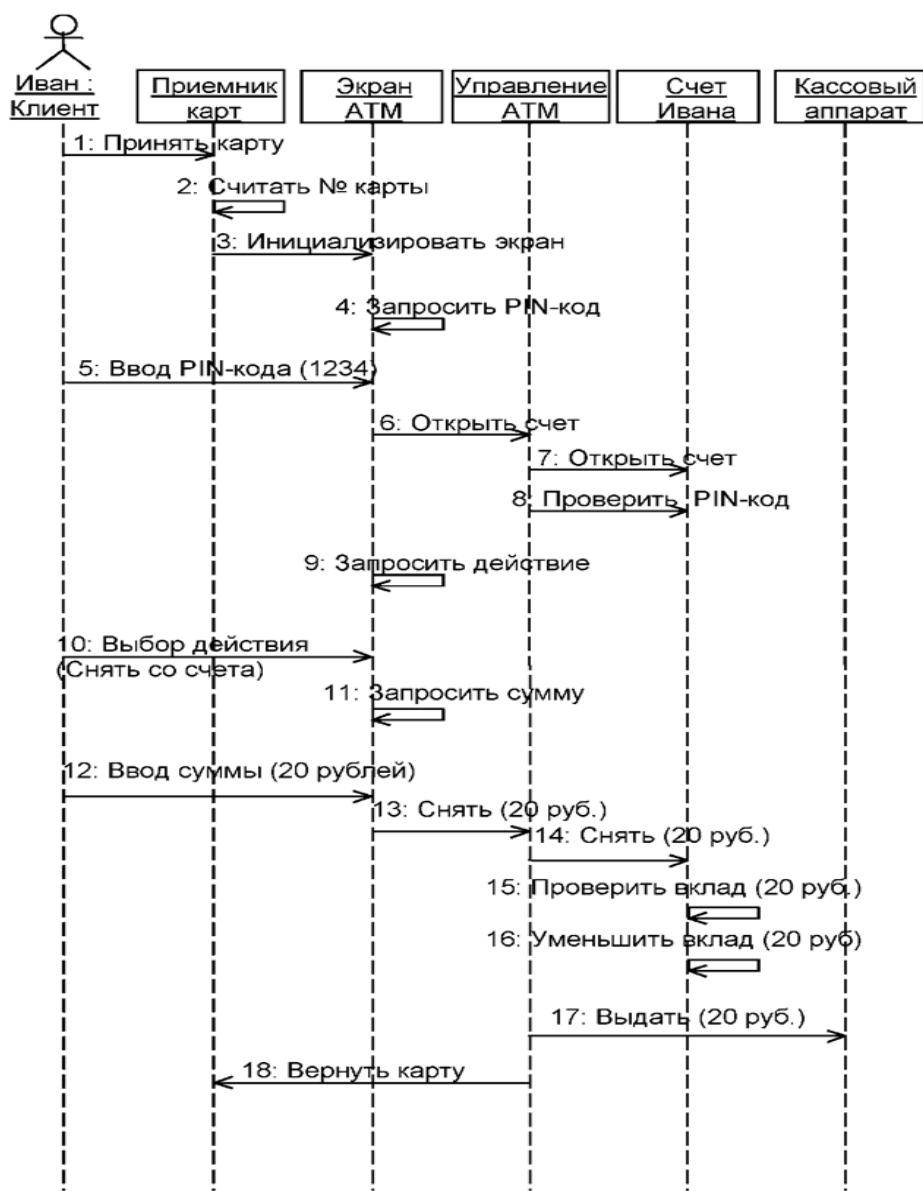


Рис. 1.4. Диаграмма последовательности для снятия клиентом денег со счета

Каждое сообщение представляется в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны на странице сверху вниз. Каждое сообщение помечается как минимум именем сообщения; при желании можно добавить также аргументы и некоторую управляющую информацию, и, кроме того, можно показать само-делегирование

(self-delegation) - сообщение, которое объект посыпает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

Хороший способ первоначального обнаружения некоторых объектов - это изучение имен существительных в потоке событий. Можно также прочитать документы, описывающие конкретный сценарий. Под сценарием понимается конкретный экземпляр потока событий. Поток событий для варианта использования «Снять деньги» говорит о человеке, снимающем некоторую сумму денег со счета с помощью ATM.

Не все объекты появляются в потоке событий. Там, например, может не быть форм для заполнения, но их необходимо показать на диаграмме, чтобы позволить действующему лицу ввести новую информацию в систему или просмотреть её. В потоке событий, скорее всего, также не будет и управляющих объектов (control objects). Эти объекты управляют последовательностью потока в варианте использования.

1.4.2. Кооперативные диаграммы

Вторым видом диаграммы взаимодействия является кооперативная диаграмма.

Подобно диаграммам последовательности, кооперативные диаграммы (collaborations) отображают поток событий через конкретный сценарий варианта использования. Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы больше внимания заостряют на связях между объектами. На рис. 1 .5 приведена кооперативная диаграмма, описывающая, как клиент снимает деньги со счета.

Как видно из рисунка, здесь представлена вся та информация, которая была и на диаграмме последовательности, но кооперативная диаграмма по-другому описывает поток событий. Из неё легче понять связи между объектами, однако, труднее уяснить последовательность событий.

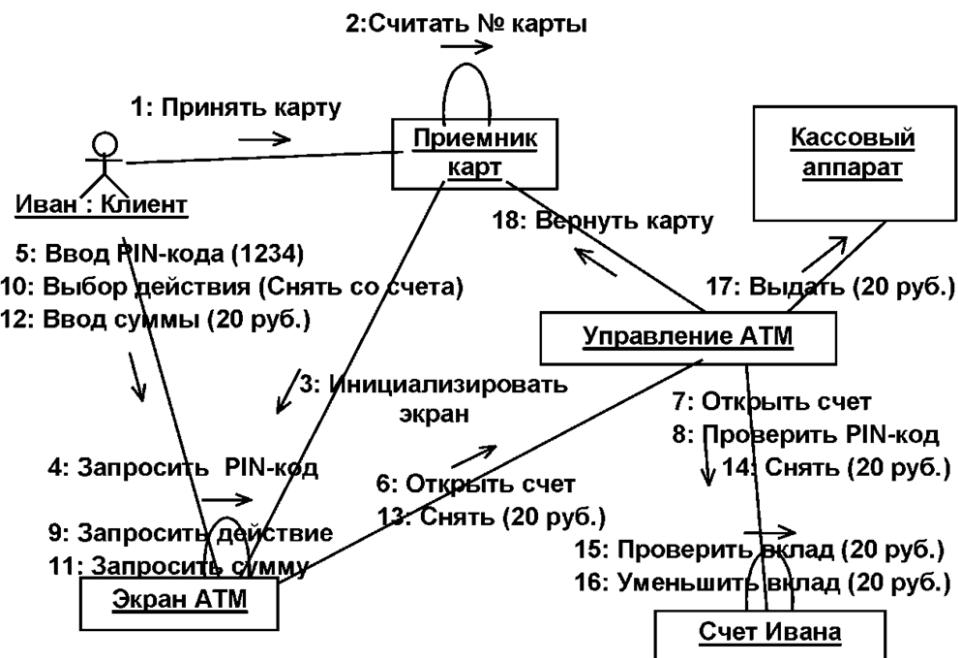


Рис. 1 .5. Кооперативная диаграмма, описывающая процесс снятия клиентом денег со своего счета

По этой причине часто для какого-либо сценария создают диаграммы обоих типов. Хотя они служат одной и той же цели и содержат одну и ту же информацию, но представляют ее с различных точек зрения.

На кооперативной диаграмме так же, как и на диаграмме последовательности, стрелки обозначают сообщения, обмен которыми осуществляется в рамках данного варианта использования. Их временная последовательность, однако, указывается путем нумерации сообщений.

1.5. Диаграммы классов

1.5.1. Общие сведения

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.

Диаграмма классов для вариантов использования «Снять деньги» показана на рис.1.6.

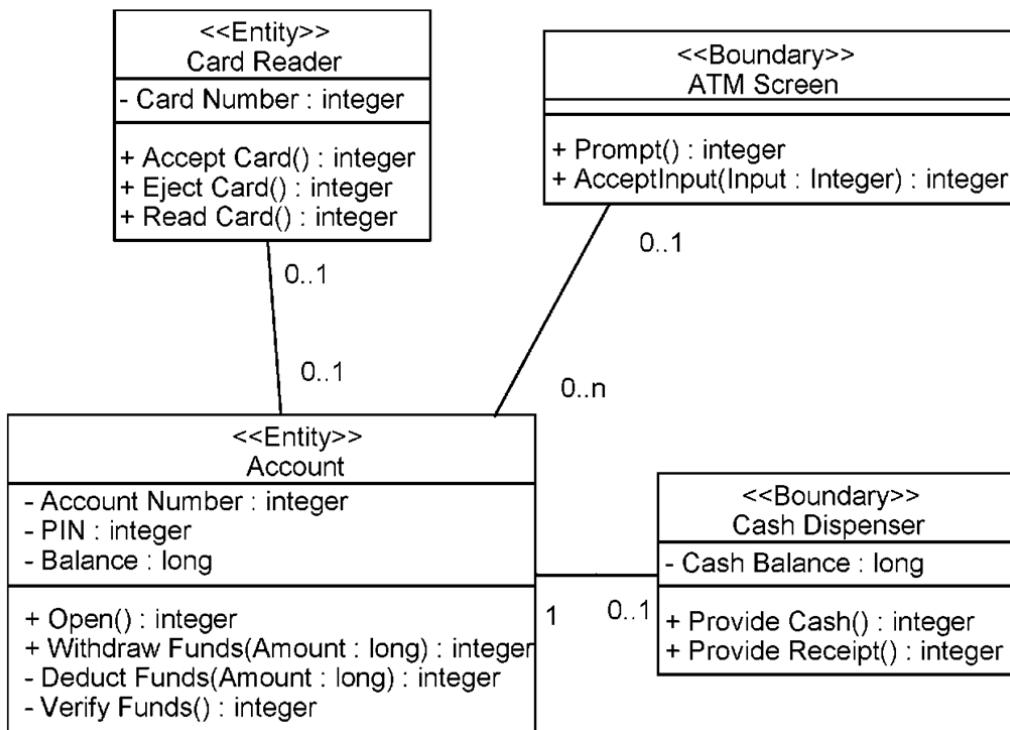


Рис. 1 .6. Диаграмма «Снять деньги

На этой диаграмме классов показаны связи между классами, реализующими вариант использования «Снять деньги». В этом процессе задействованы четыре класса: Card Reader (устройство для чтения карточек), Account (счет), ATM Screen (экран ATM) и Cash Dispenser (кассовый аппарат). Каждый класс на диаграмме выглядит в виде прямоугольника, разделенного на три части. В первой содержится имя класса, во второй - его атрибуты. В последней части содержатся операции класса, отражающие его поведение (действия, выполняемые классом).

На диаграммах классов и всех последующих диаграммах используются английские имена, так как только такие имена поддерживаются в языках программирования. Использование русских имен объектов, операций, атрибутов и т. д. сопряжено с большими трудностями, так как CASE-средства их не поддерживают должным образом.

Связывающие классы линии отражают взаимодействие между классами. Так, класс Account связан с классом ATM Screen (экран ATM), потому что они непосредственно сообщаются и взаимодействуют друг с другом. Класс Card Reader (устройство для чтения карточек) не связан с классом Cash Dispenser (кассовый аппарат), поскольку они не сообщаются друг с другом непосредственно.

1.5.2 Стереотипы классов

Стереотипы - это механизм, позволяющий разделять классы на категории. В языке UML определены три основных стереотипа классов: Boundary (граница), Entity (сущность) и Control (управление).

Границные классы

Границными классами (boundary classes) называются такие классы, которые расположены на границе системы и всей окружающей среды. Это экранные формы, отчеты, интерфейсы с аппаратурой (такой как принтеры или сканеры) и интерфейсы с другими системами.

Чтобы найти граничные классы, надо исследовать диаграммы вариантов использования. Каждому взаимодействию между действующим лицом и вариантом использования должен соответствовать, по крайней мере, один граничный класс. Именно такой класс позволяет действующему лицу взаимодействовать с системой.

Классы-сущности

Классы-сущности (entity classes) содержат хранимую информацию. Они имеют наибольшее значение для пользователя, и потому в их названиях часто используют термины из предметной области. Обычно для каждого класса-сущности создают таблицу в базе данных.

Управляющие классы

Управляющие классы (control classes) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий последовательность событий этого варианта использования. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности, так как остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посыпает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

В системе могут быть и другие управляющие классы, общие для нескольких вариантов использования. Например, может быть класс SecurityManager (менеджер безопасности), отвечающий за контроль событий, связанных с безопасностью. Класс TransactionManager (менеджер транзакций) занимается координацией сообщений, относящихся к транзакциям с базой данных. Могут быть и другие менеджеры для работы с другими элементами функционирования системы, такими как разделение ресурсов, распределенная обработка данных или обработка ошибок.

Помимо упомянутых выше стереотипов можно создавать и свои собственные.

1.5.3. Механизм пакетов

Пакеты применяют, чтобы сгруппировать классы, обладающие некоторой общностью. Существует несколько наиболее распространенных подходов к группировке. Во-первых, можно группировать их по стереотипу. В таком случае получается один пакет с классами-сущностями, один с граничными классами, один с управляющими классами и т.д. Этот подход может быть полезен с точки зрения размещения готовой системы, поскольку все находящиеся на клиентских машинах пограничные классы уже оказываются в одном пакете.

Другой подход заключается в объединении классов по их функциональности. Например, в пакете Security (безопасность) содержатся все классы, отвечающие за безопасность приложения. В таком случае другие пакеты могут называться Employee Maintenance (Работа с сотрудниками), Reporting (Подготовка отчетов) и Error Handling (Обработка ошибок). Преимущество этого подхода заключается в возможности повторного использования.

Механизм пакетов применим к любым элементам модели, а не только к классам. Если для группировки классов не использовать некоторые эвристики, то она становится произвольной. Одна из них, которая в основном используется в UML, - это зависимость. Зависимость между двумя пакетами существует в том случае, если между любыми двумя классами в пакетах существует любая зависимость. Таким образом, **диаграмма пакетов** (рис. 1 .7) представляет собой диаграмму, содержащую пакеты классов и зависимости между ними. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, то есть диаграмма пакетов - это форма диаграммы классов.

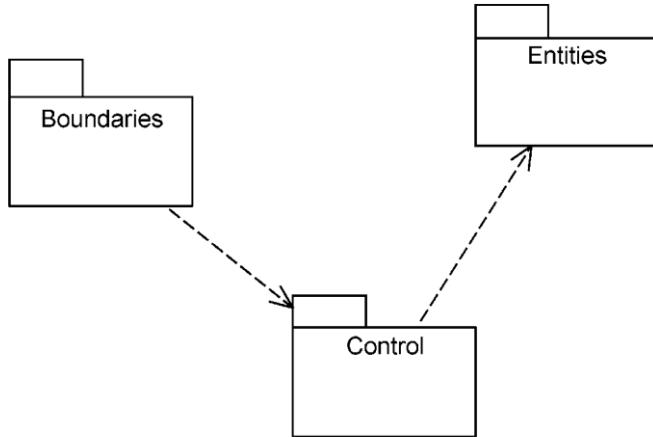


Рис. 1.7. Диаграмма пакетов

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Что касается классов, то причины для зависимостей могут быть самыми разными: один класс посыпает сообщение другому; один класс включает часть данных другого класса; один класс использует другой в качестве параметра операции. Если класс меняет свой интерфейс, то любое сообщение, которое он посыпает, может утратить свою силу.

Пакеты не дают ответа на вопрос, каким образом можно уменьшить количество зависимостей в вашей системе, однако они помогают выделить эти зависимости, а после того, как они все окажутся на виду, остается только поработать над снижением их количества. Диаграммы пакетов можно считать основным средством управления общей структурой системы.

Пакеты являются жизненно необходимым средством для больших проектов. Их следует использовать в тех случаях, когда диаграмма классов, охватывающая всю систему в целом и размещенная на единственном листе бумаги формата А4, становится нечитаемой.

1.5.4. Атрибуты

Атрибут - это элемент информации, связанный с классом. Например, у класса Company (компания) могут быть атрибуты Name (Название), Address (Адрес) и NumberOfEmployees (Число служащих).

Так как атрибуты содержатся внутри класса, они скрыты от других классов. В связи с этим может понадобиться указать, какие классы имеют право читать и изменять атрибуты. Это свойство называется видимостью атрибута (attribute visibility).

У атрибута можно определить четыре возможных значения этого параметра. Рассмотрим каждый из них в контексте примера (рис. 1.8). Пусть у нас имеется класс Employee с атрибутом Address и класс Company:

- **Public (общий, открытый).** Это значение видимости предполагает, что атрибут будет виден всеми остальными классами. Любой класс может просмотреть или изменить значение атрибута. В таком случае класс Company может изменить значение атрибута Address класса Employee. В соответствии с нотацией UML общему атрибуту предшествует знак «+».
- **Private (закрытый, секретный).** Соответствующий атрибут не виден никаким другим классом. Класс Employee будет знать значение атрибута Address и сможет изменять его, но класс Company не сможет его ни увидеть, ни редактировать. Если это понадобится, он должен попросить класс Employee просмотреть или изменить значение этого атрибута, что обычно делается с помощью общих операций. Закрытый атрибут обозначается знаком «-» в соответствии с нотацией UML.
- **Protected (защищенный).** Такой атрибут доступен только самому классу и его потомкам. Допустим, что у нас имеется два различных типа сотрудников - с почасовой оплатой и на окладе. Таким образом, мы получаем два других класса HourlyEmp и SalariedEmp, являющихся потомками класса Employee. Защищенный атрибут Address можно просмотреть

или изменить из классов Employee, HourlyEmp и SalariedEmp, но не из класса Company. Нотация UML для защищенного атрибута - это знак «#». - **Package or Implementation (пакетный)**. Предполагает, что данный атрибут является общим, но только в пределах его пакета. Допустим, что атрибут Address имеет пакетную видимость. В таком случае он может быть изменен из класса Company, только если этот класс находится в том же пакете. Этот тип видимости не обозначается никаким специальным значком.

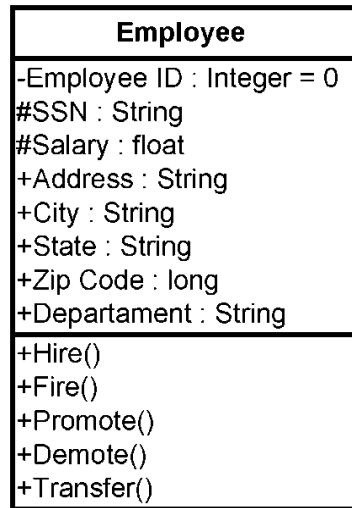


Рис. 1.8. Видимость атрибутов

В общем случае, атрибуты рекомендуется делать закрытыми или защищенными. Это позволяет лучше контролировать сам атрибут и код. С помощью закрытости или защищенности удается избежать ситуации, когда значение атрибута изменяется всеми классами системы. Вместо этого логика изменения атрибута будет заключена в том же классе, что и сам этот атрибут. Задаваемые параметры видимости повлияют на генерируемый код.

1.5.5. Операции

Операции реализуют связанное с классом поведение. Операция включает три части - имя, параметры и тип возвращаемого значения. Параметры - это аргументы, получаемые операцией «на входе». Тип возвращаемого значения относится к результату действия операции.

На диаграмме классов можно показывать как имена операций, так и имена операций вместе с их параметрами и типом возвращаемого значения. Чтобы уменьшить загруженность диаграммы, полезно бывает на некоторых из них показывать только имена операций, а на других их полную сигнатуру.

В языке UML операции имеют следующую нотацию:

Имя Операции (аргумент₁ : тип данных аргумент₁, аргумент₂ : тип данных аргумент₂,...) : тип возвращаемого значения

Следует рассмотреть четыре различных типа операций.

Операции реализации

Операции реализации (implementor operations) реализуют некоторые бизнес-функции. Такие операции можно найти, исследуя диаграммы взаимодействия. Диаграммы этого типа фокусируются на бизнес-функциях, и каждое сообщение диаграммы, скорее всего, можно соотнести с операцией реализации.

Каждая операция реализации должна быть легко прослеживаема до соответствующего требования. Это достигается на различных этапах моделирования. Операция выводится из сообщения на диаграмме взаимодействия, сообщения исходят из подробного описания потока событий, который создается на основе варианта использования, а последний - на основе требований. Возможность проследить всю эту цепочку позволяет гарантировать, что каждое требование будет реализовано в коде, а каждый фрагмент кода реализует какое-то требование.

Операции управления

Операции управления (manager operations) управляют созданием и уничтожением объектов. В эту категорию попадают конструкторы и деструкторы классов.

Операции доступа

Атрибуты обычно бывают закрытыми или защищенными. Тем не менее, другие классы иногда должны просматривать или изменять их значения. Для этого существуют операции доступа (access operations).

Пусть, например, у нас имеется атрибут Salary класса Employee. Мы не хотим, чтобы все остальные классы могли изменять этот атрибут. Вместо этого к классу Employee мы добавляем две операции доступа - GetSalary и SetSalary. К первой из них, являющейся общей, могут обращаться и другие классы. Она просто получает значение атрибута Salary и возвращает его вызвавшему ее классу. Операция SetSalary также является общей, она помогает вызвавшему ее классу установить новое значение атрибута Salary. Эта операция может содержать любые правила и условия проверки, которые необходимо выполнить, чтобы зарплата могла быть изменена.

Такой подход дает возможность безопасно инкапсулировать атрибуты внутри класса, защитив их от других классов, но все же позволяет осуществить к ним контролируемый доступ. Создание операций Get и Set (получения и изменения значения) для каждого атрибута класса является стандартом.

Вспомогательные операции

Вспомогательными (helper operations) называются такие операции класса, которые необходимы ему для выполнения его ответственостей, но о которых другие классы не должны ничего знать. Это закрытые и защищенные операции класса.

Чтобы идентифицировать операции, выполните следующие действия:

1. Изучите диаграммы последовательности и кооперативные диаграммы. Большая часть сообщений на этих диаграммах является операциями реализации. Рефлексивные сообщения будут вспомогательными операциями.
2. Рассмотрите управляющие операции. Может потребоваться добавить конструкторы и деструкторы.
3. Рассмотрите операции доступа. Для каждого атрибута класса, с которым должны будут работать другие классы, надо создать операции Get и Set.

1.5.6. Связи

Связь представляет собой семантическую взаимосвязь между классами. Она дает классу возможность узнавать об атрибутах, операциях и связях другого класса. Иными словами, чтобы один класс мог послать сообщение другому на диаграмме последовательности или кооперативной диаграмме, между ними должна существовать связь.

Существуют четыре типа связей, которые могут быть установлены между классами: ассоциации, зависимости, агрегации и обобщения.

Ассоциации

Ассоциация (association) - это семантическая связь между классами. Их рисуют на диаграмме классов в виде обычновенной линии.



Рис. 1 .9. Ассоциация

Ассоциации могут быть двунаправленными, как в примере, или однонаправленными. На языке UML двунаправленные ассоциации рисуют в виде простой линии без стрелок или со стрелками с обеих ее сторон. На однонаправленной ассоциации изображают только одну стрелку, показывающую ее направление.

Направление ассоциации можно определить, изучая диаграммы последовательности и кооперативные диаграммы. Если все сообщения на них отправляются только одним классом и принимаются только другим классом, но не наоборот, между этими классами имеет место односторонняя связь. Если хотя бы одно сообщение отправляется в обратную сторону, ассоциация должна быть двунаправленной.

Ассоциации могут быть рефлексивными. Рефлексивная ассоциация предполагает, что один экземпляр класса взаимодействует с другими экземплярами этого же класса.

Зависимости

Связи зависимости (dependency) также отражают связь между классами, но они всегда односторонны и показывают, что один класс зависит от определений, сделанных в другом. Зависимости изображают в виде стрелки, проведенной пунктирной линией.



Рис. 1.10. Зависимость

При генерации кода для этих классов к ним не будут добавляться новые атрибуты. Однако, будут созданы специфические для языка операторы, необходимые для поддержки связи. Например, на языке C++ в код войдут необходимые операторы #include.

Агрегации

Агрегации (aggregations) представляют собой более тесную форму ассоциации. Агрегация - это связь между целым и его частью. Например, у вас может быть класс Автомобиль, а также классы Двигатель, Покрышки и классы для других частей автомобиля. В результате объект класса Автомобиль будет состоять из объекта класса Двигатель, четырех объектов Покрышек и т.д. Агрегации визуализируют в виде линии с ромбиком у класса, являющегося целым:

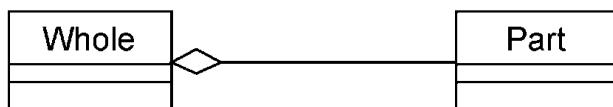


Рис. 1.11. Агрегация на его части.

В дополнение к простой агрегации UML вводит более сильную разновидность агрегации, называемую композицией. Согласно композиции, объект-часть может принадлежать только единственному целому, и, кроме того, как правило, жизненный цикл частей совпадает с циклом целого: они живут и умирают вместе с ним. Любое удаление целого распространяется на его части.

Такое каскадное удаление нередко рассматривается как часть определения агрегации, однако оно всегда подразумевается в том случае, когда множественность роли составляет 1:1, например, если необходимо удалить Клиента, то это удаление должно распространиться и на Заказы (и, в свою очередь, на Строки заказа).

Обобщения

С помощью обобщений (generalization) показывают связи наследования между двумя классами. Большинство объектно-ориентированных языков непосредственно поддерживают концепцию наследования. Она позволяет одному классу наследовать все атрибуты, операции и связи другого. На языке UML связи наследования называют обобщениями и изображают в виде стрелок от класса-потомка к классу-предку:

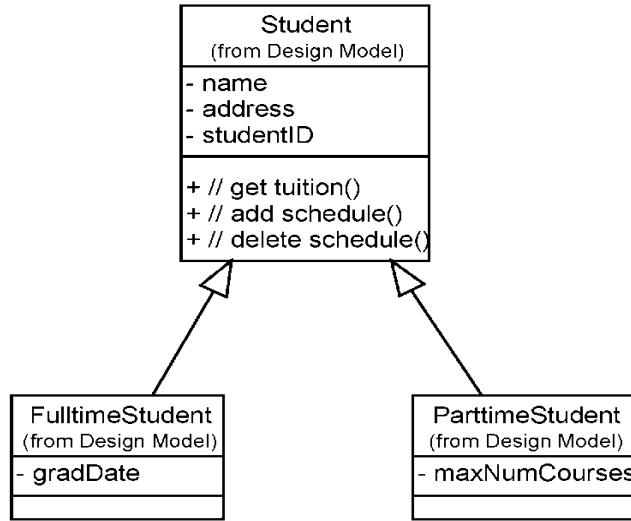


Рис. 1.12. Обобщение

Помимо наследуемых, каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.

Множественность

Множественность (multiplicity) показывает, сколько экземпляров одного класса взаимодействуют с помощью этой связи с одним экземпляром другого класса в данный момент времени.

Например, при разработке системы регистрации курсов в университете можно определить классы Course (курс) и Student (студент). Между ними установлена связь: у курсов могут быть студенты, а у студентов - курсы. Вопросы, на который должен ответить параметр множественности: «Сколько курсов студент может посещать в данный момент? Сколько студентов может за раз посещать один курс?» Так как множественность дает ответ на оба эти вопроса, её индикаторы устанавливаются на обоих концах линии связи. В примере регистрации курсов мы решили, что один студент может посещать от нуля до четырех курсов, а один курс могут слушать от 10 до 20 студентов. На диаграмме классов это можно изобразить, как показано на рис. 1.13.

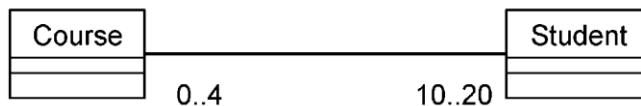


Рис. 1.13. Множественность

В языке UML приняты следующие нотации для обозначения множественности:

Множественность	Значение
0..*	Ноль или больше
1..*	Один или больше
0..1	Ноль или один
1..1 (сокращенная запись: 1)	Ровно один

Имена связей

Связи можно уточнить с помощью имен связей или ролевых имен. Имя связи - это обычно глагол или глагольная фраза, описывающая, зачем она нужна. Например, между классом Person (человек) и классом Company (компания) может существовать ассоциация. Можно задать в связи с этим вопрос, является ли объект класса Person клиентом компании, её сотрудником или владельцем? Чтобы определить это, ассоциацию можно назвать «employs» (нанимает):



Рис. 1.14. Имя связи

Имена у связей определять не обязательно. Обычно это делают, если причина создания связи не очевидна. Имя показывают около линии соответствующей связи.

Роли

Ролевые имена применяют в связях ассоциации или агрегации вместо имен для описания того, зачем эти связи нужны. Возвращаясь к примеру с классами Person и Company, можно сказать, что класс Person играет роль сотрудника класса Company. Ролевые имена - это обычно имена существительные или основанные на них фразы, их показывают на диаграмме рядом с классом, играющим соответствующую роль. Как правило, пользуются или ролевым именем, или именем связи, но не обоими сразу. Как и имена связей, ролевые имена не обязательны, их дают, только если цель связи не очевидна. Пример ролей приводится ниже:

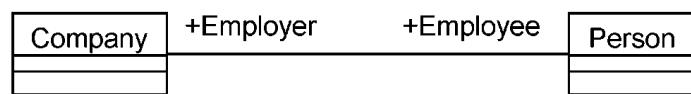


Рис. 1.15. Ролевые имена

1.6. Диаграммы состояний

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой. Наиболее распространенная форма, используемая в объектно-ориентированных методах, впервые применялась в методе ОМТ и впоследствии была адаптирована Гради Бучем.

На рис. 1.16 приводится пример диаграммы состояний для банковского счета. Из данной диаграммы видно, в каких состояниях может существовать счет. Можно также видеть процесс перехода счета из одного состояния в другое. Например, если клиент требует закрыть открытый счет, он переходит в состояние «Закрыт». Требование клиента называется событием (event), именно такие события и вызывают переход из одного состояния в другое.

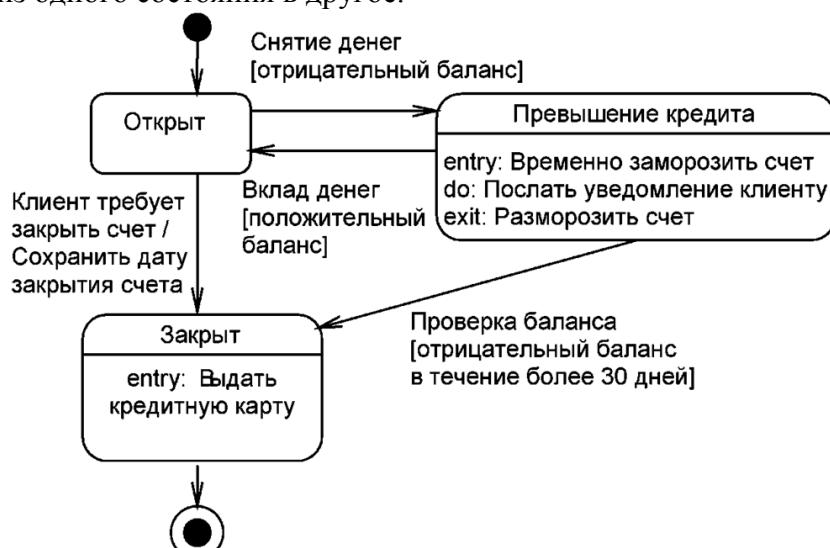


Рис. 1.16. Диаграмма состояний для класса Account

Если клиент снимает деньги с открытого счета, он может перейти в состояние «Превышение кредита». Это происходит, только если баланс по этому счету меньше ноля, что отражено условием

[отрицательный баланс] на нашей диаграмме. Заключенное в квадратных скобках условие (guard condition) определяет, когда может или не может произойти переход из одного состояния в другое.

На диаграмме имеются два специальных состояния - начальное (start) и конечное (stop). Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. На диаграмме состояний может быть одно и только одно начальное состояние. В то же время, может быть столько конечных состояний, сколько вам нужно, или их может не быть вообще. Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. В нашем примере при превышении кредита клиенту посыпается соответствующее сообщение. Процессы, происходящие, когда объект находится в определенном состоянии, называются действиями (actions).

С состоянием можно связывать данные пяти типов: деятельность, входное действие, выходное действие, событие и история состояния. Рассмотрим каждый из них в контексте диаграммы состояний для класса Account системы ATM.

Деятельность

Деятельностью (activity) называется поведение, реализуемое объектом, пока он находится в данном состоянии. Например, когда счет находится в состоянии «Закрыт», происходит возврат кредитной карточки пользователю. Деятельность - это прерываемое поведение. Оно может выполняться до своего завершения, пока объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

Входное действие

Входным действием (entry action) называется поведение, которое выполняется, когда объект переходит в данное состояние. В примере счета в банке, когда он переходит в состояние «Превышен счет», выполняется действие «Временно заморозить счет», независимо от того, откуда объект перешел в это состояние. Таким образом, данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть этого перехода. В отличие от деятельности, входное действие рассматривается как непрерываемое.

Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

Выходное действие

Выходное действие (exit action) подобно входному. Однако, оно осуществляется как составная часть процесса выхода из данного состояния. В нашем примере при выходе объекта Account из состояния «Превышен счет», независимо от того, куда он переходит, выполняется действие «Разморозить счет». Оно является частью процесса такого перехода. Как и входное, выходное действие является непрерываемым.

Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Поведение объекта во время деятельности, при входных и выходных действиях может включать отправку события другому объекту. Например, объект account (счет) может посыпать событие объекту card reader (устройство чтения карты). В этом случае описанию деятельности, входного действия или выходного действия предшествует знак « ^ ». Соответствующая строка на диаграмме выглядит как

Do: ^ Цель. Событие(Аргументы)

Здесь Цель - это объект, получающий событие, Событие - это посыпаемое сообщение, а Аргументы являются параметрами посыпаемого сообщения.

Деятельность может также выполняться в результате получения объектом некоторого события. Например, объект account может быть в состоянии Открыто. При получении некоторого события выполняется определенная деятельность.

Переходом (Transition) называется перемещение из одного состояния в другое. Совокупность переходов диаграммы показывает, как объект может перемещаться между своими состояниями. На

диаграмме все переходы изображают в виде стрелки, начинающейся на первоначальном состоянии и заканчивающейся последующим.

Переходы могут быть рефлексивными. Объект может перейти в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии.

У перехода существует несколько спецификаций. Они включают события, аргументы, ограждающие условия, действия и посылаемые события. Рассмотрим каждое из них в контексте примера ATM.

События

Событие (event) - это то, что вызывает переход из одного состояния в другое. В нашем примере событие «Клиент требует закрыть» вызывает переход счета из открытого в закрытое состояние. События размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции, так и обычную фразу. В нашем примере события описаны обычными фразами. Если вы хотите использовать операции, то событие «Клиент требует закрыть» можно было бы назвать RequestClosure().

У событий могут быть аргументы. Так, событие «Сделать вклад», вызывающее переход счета из состояния «Превышен счет» в состояние «Открыт», может иметь аргумент Amount (Количество), описывающий сумму депозита.

Большинство переходов должны иметь события, так как именно они, прежде всего, заставляют переход осуществляться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществляться входным действиям, деятельности и выходным действиям.

Ограждающие условия

Ограждающие условия (guard conditions) определяют, когда переход может, а когда не может осуществляться. В нашем примере событие «Сделать вклад» переведет счет из состояния «Превышение счета» в состояние «Открыт», но только если баланс будет больше нуля. В противном случае переход не осуществляется.

Ограждающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки.

Ограждающие условия задавать необязательно. Однако если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет автоматически выбран.

Действие

Действием (action), как уже говорилось, является непрерываемое поведение, осуществляющееся как часть перехода. Входные и выходные действия показывают внутри состояний, поскольку они определяют, что происходит, когда объект входит или выходит из него. Большую часть действий, однако, изображают вдоль линии перехода, так как они не должны осуществляться при входе или выходе из состояния.

Например, при переходе счета из открытого в закрытое состояние выполняется действие «Сохранить дату закрытия счета». Это непрерываемое поведение осуществляется только во время перехода из состояния «Открыт» в состояние «Закрыт».

Действие рисуют вдоль линии перехода после имени события, ему предшествует косая черта.

Событие или действие могут быть поведением внутри объекта, а могут представлять собой сообщение, посылаемое другому объекту. Если событие или действие посыпается другому объекту, перед ним на диаграмме помещают знак « А ».

Диаграммы состояний не надо создавать для каждого класса, они применяются только в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, для него может потребоваться такая диаграмма.

1.7. Диаграммы деятельности

В отличие от большинства других средств UML, диаграммы деятельности не имеют явно выраженного источника в предыдущих работах Буча, Рамбо и Якобсона, и заимствуют идеи из нескольких различных методов, в частности, метода моделирования состояний SDL и сетей Петри. Эти диаграммы особенно полезны в описании поведения, включающего большое количество параллельных процессов [Фаулер-1999].

Подобно большинству других средств, моделирующих поведение, диаграммы деятельности обладают определенными достоинствами и недостатками, поэтому их лучше всего использовать в сочетании с другими средствами.

Самым большим достоинством диаграмм деятельности является поддержка параллелизма. Благодаря этому они являются мощным средством моделирования потоков работ и, по существу, параллельного программирования. Самый большой их недостаток заключается в том, что связи между действиями и объектами просматриваются не слишком четко.

Эти связи можно попытаться определить, используя для действий метки с именами объектов, но этот способ не обладает такой же простой непосредственностью, как у диаграмм взаимодействия. Диаграммы деятельности предпочтительнее использовать в следующих ситуациях:

- Анализ варианта использования. На этой стадии нас не интересует связь между действиями и объектами, а нужно только понять, какие действия должны иметь место и каковы зависимости в поведении системы. Связывание методов и объектов выполняется позднее с помощью диаграмм взаимодействия.
- Анализ потоков работ (workflow) в различных вариантах использования. Когда варианты использования взаимодействуют друг с другом, диаграммы деятельности являются мощным средством представления и анализа их поведения.

1.8. Диаграммы компонентов

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. После создания они сразу добавляются к диаграмме компонентов. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы.

На рис. 1.17 изображена одна из диаграмм компонентов для системы ATM.

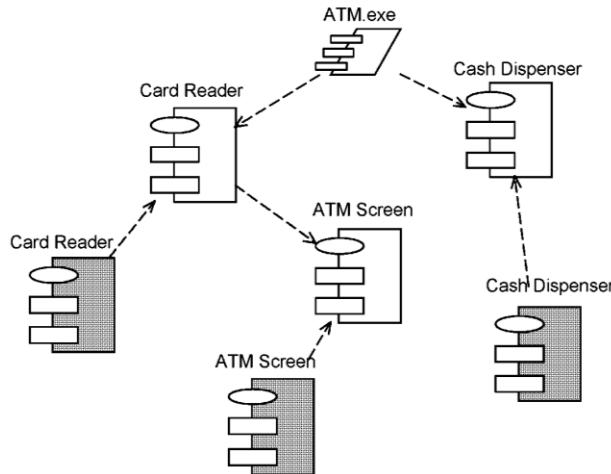


Рис. 1.17. Диаграмма компонентов для клиента ATM

На этой диаграмме показаны компоненты клиента системы ATM. В данном случае система разрабатывается на языке C++. У каждого класса имеется свой собственный заголовочный файл и файл с расширением .CPP, так что каждый класс преобразуется в свои собственные компоненты на диаграмме. Например, класс ATM screen преобразуется в компонент ATM Screen диаграммы. Он

преобразуется также и во второй компонент ATM Screen. Вместе эти два компонента представляют тело и заголовок класса ATM Screen. Выделенный темным компонент называется спецификацией пакета (package specification) и соответствует файлу тела класса ATM Screen на языке C++ (файл с расширением .CPP). Невыделенный компонент также называется спецификацией пакета, но соответствует заголовочному файлу класса языка C++ (файл с расширением .H). Компонент ATM.exe является спецификацией задачи и представляет поток обработки информации (thread of processing). В данном случае поток обработки является исполняемой программой.

Компоненты соединены штриховой линией, что соответствует зависимостям между ними. Например, класс Card Reader зависит от класса ATM Screen. Это означает, что, для того, чтобы класс Card Reader мог быть скомпилирован, класс ATM Screen должен уже существовать. После компиляции всех классов может быть создан исполняемый файл ATMClient.exe.

Пример ATM содержит два потока обработки и, таким образом, получаются два исполняемых файла. Один из них - это клиент ATM, он содержит компоненты Cash Dispenser, Card Reader и ATM Screen. Второй файл - это сервер ATM, включающий в себя компонент Account. Диаграмма компонентов для сервера ATM показана на рис. 1.18.

Как видно из примера, у системы может быть несколько диаграмм компонентов, в зависимости от числа подсистем или исполняемых файлов. Каждая подсистема является пакетом компонентов. В общем случае, пакеты - это совокупности компонентов. Пример ATM содержит два пакета: клиент ATM и сервер ATM.

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы. Из нее видно, в каком порядке надо компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. На такой диаграмме показано соответствие классов реализованным компонентам. Она нужна там, где начинается генерация кода.

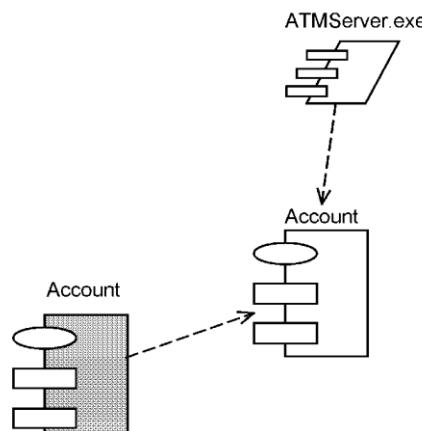


Рис. 1.18. Диаграмма Компонентов для сервера ATM

1.9. Диаграммы размещения

Диаграмма размещения (deployment diagram) отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать маршруты перемещения объектов и компонентов в распределенной системе.

Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства - в большинстве случаев, часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком, а может быть и майнфреймом.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов. В нашем примере система ATM состоит из большого количества подсистем, выполняемых на отдельных физических устройствах, или узлах (node). Диаграмма размещения для системы ATM показана на рис. 1.19.

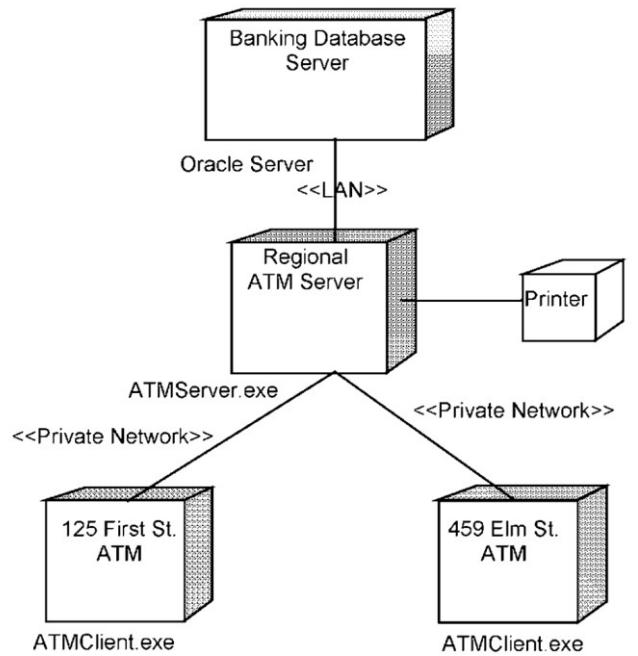


Рис. 1.19. Диаграмма размещения для системы АТМ

Из данной диаграммы можно узнать о физическом размещении системы. Клиентские программы АТМ будут работать в нескольких местах на различных сайтах. Через закрытые сети будет осуществляться их сообщение с региональным сервером АТМ. На нём будет работать программное обеспечение сервера АТМ. В свою очередь, посредством локальной сети региональный сервер будет сообщаться с сервером банковской базы данных, работающим под управлением Oracle. Наконец, с региональным сервером АТМ соединен принтер.

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение её отдельных подсистем

Глава 2. Основные сведения о CASE-средстве Rational Rose

*Некоторые проектные команды рассматривают CASE-средства как „костыли“ для новичков, а другие считают их не менее важными, чем текстовые процессоры.
Э. Йордон „Путь камикадзе“*

2.1. Введение в Rational Rose

Rational Rose - семейство объектно-ориентированных CASE-средств фирмы Rational Software Corporation - предназначено для автоматизации процессов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose использует метод объектно-ориентированного анализа и проектирования, основанный на языке UML. Текущая версия Rational Rose реализует генерацию кодов программ для C++, Visual C++, Visual Basic, Java, PowerBuilder, CORBA Interface Definition Language (IDL), генерацию описаний баз данных для ANSI SQL, Oracle, MS SQL Server, IBM DB2, Sybase, а также позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций. Кроме того, Rational Rose содержит средства реверсного инжиниринга программ и баз данных, обеспечивающие повторное использование программных компонентов в новых проектах.

Структура и функции. В основе работы Rational Rose лежит построение диаграмм и спецификаций UML, определяющих архитектуру системы, её статические и динамические аспекты. В составе Rational Rose можно выделить шесть основных структурных компонентов: репозиторий, графический интерфейс пользователя, средства просмотра проекта (браузер), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генератор кодов (индивидуальный для каждого языка) и анализатор для C++, обеспечивающий реверсный инжиниринг.

Репозиторий представляет собой базу данных проекта. Браузер обеспечивает "навигацию" по проекту, в том числе перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т. д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по мере развития проекта, а не после завершения его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кодов программ на языке C++, используя информацию, содержащуюся в диаграммах классов и компонентов, формируют файлы заголовков и файлы описаний классов и объектов. Создаваемый таким образом скелет программы может быть уточнен путем прямого программирования на языке C++. Анализатор кодов C++ реализован в виде отдельного программного модуля. Его назначение - создавать модули проектов Rational Rose на основе информации, содержащейся в определяемых пользователем исходных текстах на C++. В процессе работы анализатор осуществляет контроль правильности исходных текстов и диагностику ошибок. Модель, полученная в результате его работы, может целиком или фрагментарно использоваться в различных проектах. Анализатор обладает широкими возможностями настройки по входу и выходу. Например, можно определить типы исходных файлов, базовый компилятор, задать, какая информация должна быть включена в формируемую модель, и какие элементы выходной модели следует выводить на экран. Таким образом, Rational Rose/C++ обеспечивает возможность повторного использования программных компонентов.

В результате разработки проекта с помощью CASE-средства Rational Rose формируются следующие документы:

- диаграммы UML, в совокупности представляющие собой модель разрабатываемой программной системы;
- спецификации классов, объектов, атрибутов и операций;
- заготовки текстов программ.

Тексты программ являются заготовками для последующей работы программистов. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

Взаимодействие с другими средствами и организация групповой работы. Для поддержки командной работы над проектом на каждой стадии жизненного цикла ПО имеется интегрированный набор продуктов Rational Suite. Rational Suite существует в следующих вариантах:

- Rational Suite AnalystStudio - предназначен для определения и управления полным набором требований к разрабатываемой системе;
- Rational Suite DevelopmentStudio - предназначен для проектирования и реализации ПО;
- Rational Suite TestStudio - представляет собой набор продуктов, предназначенных для автоматического тестирования приложений;
- Rational Suite Enterprise - обеспечивает поддержку полного жизненного цикла ПО и предназначен как для менеджеров проекта, так и отдельных разработчиков, выполняющих несколько функциональных ролей в команде разработчиков.

В состав Rational Suite, кроме Rational Rose, входят следующие компоненты:

- Rational Requisite Pro - средство управления требованиями, предназначенное для организации совместной работы группы разработчиков. Оно позволяет команде разработчиков создавать, структурировать, устанавливать приоритеты, отслеживать, контролировать изменения требований, возникающих на любом этапе разработки компонентов приложения;
- Rational ClearCase - средство управления конфигурацией ПО;
- Rational SoDA - средство автоматической генерации проектной документации;
- Rational ClearQuest - средство для управления изменениями и отслеживания дефектов в проекте на основе средств e-mail и Web;
- Rational TeamTest - средство автоматического обнаружения ошибок во время выполнения программы и генерации сценариев для проведения регрессионного тестирования;
- Rational Robot - средство для создания, модификации и автоматического запуска тестов;
- Rational Purify - средство для локализации трудно обнаруживаемых ошибок времени выполнения программы;
- Rational PureCoverage - средство идентификации участков кода, пропущенных при тестировании;
- Rational Quantify - средство количественного определения узких мест, влияющих на общую эффективность работы программы;
- Rational Suite PerformanceStudio - средство нагрузочного тестирования приложений «клиент-сервер» и Web-приложений.

Для организации групповой работы в Rational Rose возможно разбиение модели на управляемые подмодели. Каждая из них независимо сохраняется на диске или загружается в модель. В качестве подмодели может выступать пакет или подсистема.

Среда функционирования. Rational Rose функционирует на различных платформах: IBM PC (Windows 95/98/NT), Sun SPARCstations (UNIX, Solaris, SunOS), Hewlett-Packard (HP UX), IBM RS/6000 (AIX).

2.2. Работа в среде Rational Rose

2.2.1. Элементы экрана

Пять основных элементов интерфейса Rose - это браузер, окно документации, панели инструментов, окно диаграммы и журнал (log). Их назначение заключается в следующем:

- браузер (browser) - используется для быстрой навигации по модели;
- окно документации (documentation window) - применяется для работы с текстовым описанием элементов модели;
- панели инструментов (toolbars) - применяются для быстрого доступа к наиболее распространенным командам;
- окно диаграммы (diagram window) - используется для просмотра и редактирования одной или нескольких диаграмм UML;
- журнал (log) - применяется для просмотра ошибок и отчетов о результатах выполнения различных команд.

На рис. 2. 1 показаны различные части интерфейса Rose.

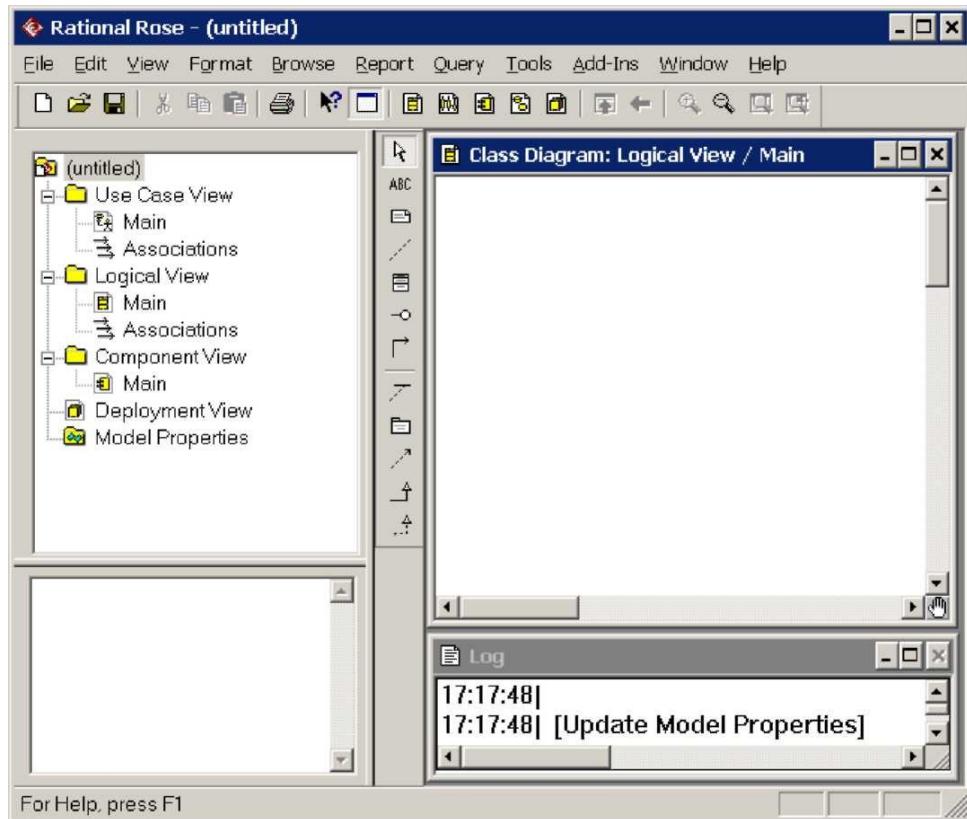


Рис. 2.1. Интерфейс Rational Rose.

Браузер

Браузер - это иерархическая структура, позволяющая осуществлять навигацию по модели. Все, что добавляется в модель - действующие лица, варианты использования, классы, компоненты - будет показано в окне браузера. С помощью браузера можно:

- добавлять в модель элементы (действующие лица, варианты использования, классы, компоненты, диаграммы и т.д.);
- просматривать существующие элементы модели;
- просматривать существующие связи между элементами модели;
- перемещать элементы модели;
- переименовывать эти элементы;
- добавлять элементы модели к диаграмме;
- связывать элемент с файлом или адресом Интернет;
- группировать элементы в пакеты;
- работать с детализированной спецификацией элемента;
- открывать диаграмму.

Браузер поддерживает четыре представления (view): представление вариантов использования, компонентов, размещения и логическое представление. Все они и содержащиеся в них элементы модели описаны ниже в подразд. 2.2.2.

Браузер организован в древовидном стиле. Каждый элемент модели может содержать другие элементы, находящиеся ниже его в иерархии. Знак «-» около элемента означает, что его ветвь полностью раскрыта. Знак «+» - что его ветвь свернута.

Окно документации

С его помощью можно документировать элементы модели Rose. Например, можно сделать короткое описание каждого действующего лица. При документировании класса все, что будет написано в окне документации, появится затем в виде комментария в сгенерированном коде, что

избавляет от необходимости впоследствии вносить эти комментарии вручную. Документация будет выводиться также в отчетах, создаваемых в среде Rose.

Панели инструментов

Панели инструментов Rose обеспечивают быстрый доступ к наиболее распространенным командам. В этой среде существует два типа панелей инструментов: стандартная панель и панель диаграммы. Стандартная панель видна всегда, ее кнопки соответствуют командам, которые могут использоваться для работы с любой диаграммой. Панель диаграммы своя для каждого типа диаграмм UML.

Все панели инструментов могут быть изменены и настроены пользователем. Для этого выберите пункт меню Tools > Options, затем выберите вкладку Toolbars.

Чтобы показать или скрыть стандартную панель инструментов (или панель инструментов диаграммы):

- 1 . Выберите пункт Tools > Options.
2. Выберите вкладку Toolbars.
3. Чтобы сделать видимой или невидимой стандартную панель инструментов, пометьте (или снимите пометку) контрольный переключатель Show StandardToolBar (или Show DiagramToolBar)

Чтобы увеличить размер кнопок на панели инструментов:

1. Щелкните правой кнопкой мыши на требуемой панели.
2. Выберите во всплывающем меню пункт Use Large Buttons (Использовать большие кнопки)

Чтобы настроить панель инструментов:

- 1 . Щелкните правой кнопкой мыши на требуемой панели.
2. Выберите пункт Customize (настроить)
3. Чтобы добавить или удалить кнопки, выберите соответствующую кнопку и затем щелкните мышью на кнопке Add (добавить) или Remove (удалить), как показано на рис. 2.2.

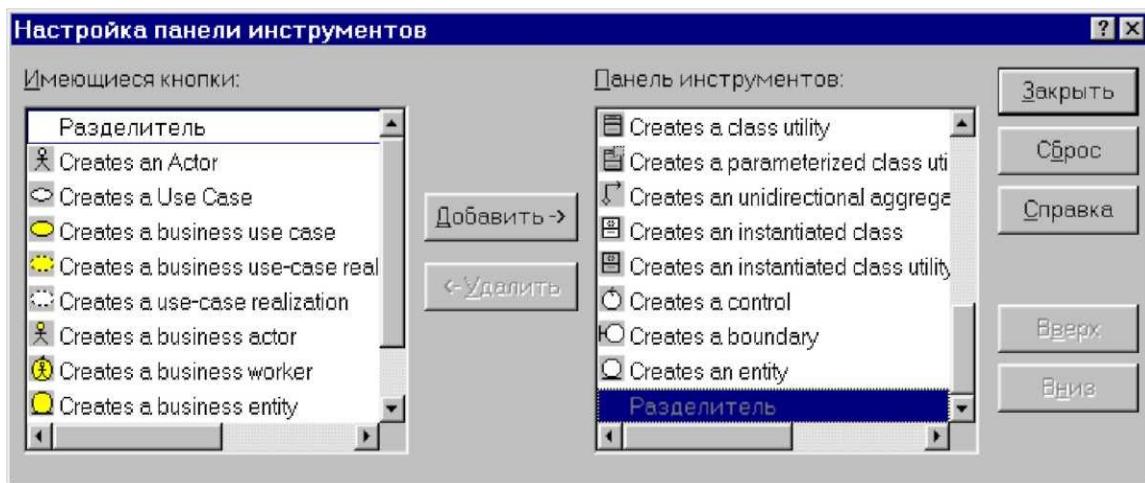


Рис. 2.2. Настройка стандартной панели инструментов. **Окно диаграммы**

В окне диаграммы видно, как выглядит одна или несколько диаграмм UML модели. При внесении в элементы диаграммы изменений Rose автоматически обновит браузер. Аналогично, при внесении изменений в элемент с помощью браузера Rose автоматически обновит соответствующие диаграммы. Это помогает поддерживать модель в непротиворечивом состоянии.

Журнал

По мере работы над вашей моделью определенная информация будет направляться в окно журнала. Например, туда помещаются сообщения об ошибках, возникающих при генерации кода. Не существует способа закрыть журнал совсем, но его окно может быть минимизировано.

2.2.2. Четыре представления модели Rose

В модели Rose поддерживается четыре представления (views) -представление вариантов использования, логическое представление, представление компонентов и представление размещения. Каждое из них предназначено для своих целей и для соответствующей аудитории. В последующих разделах этой главы мы кратко рассмотрим каждое из указанных представлений, а в оставшейся части книги детально обсудим содержащиеся в них элементы модели.

Представление вариантов использования

Это представление содержит всех действующих лиц, все варианты использования и их диаграммы для конкретной системы. Оно может также содержать некоторые диаграммы последовательности и кооперативные диаграммы. На рис. 2.3 показано, как выглядит представление вариантов использования в браузере Rose.

Представление вариантов использования содержит:

- Действующих лиц.
- Варианты использования.
- Документацию по вариантам использования, детализирующую происходящие в них процессы (потоки событий), включая обработку ошибок. Пиктограммами изображаются внешние файлы, прикрепленные к модели Rose. Вид пиктограммы, зависит от приложения, используемого для документирования потока событий. В данном случае (рис. 2.3) применялся Microsoft Word.
- Диаграммы вариантов использования. Обычно у системы бывает несколько таких диаграмм, каждая из которых показывает подмножество действующих лиц и/или вариантов использования.
- Пакеты, являющиеся группами вариантов использования и/или действующих лиц.

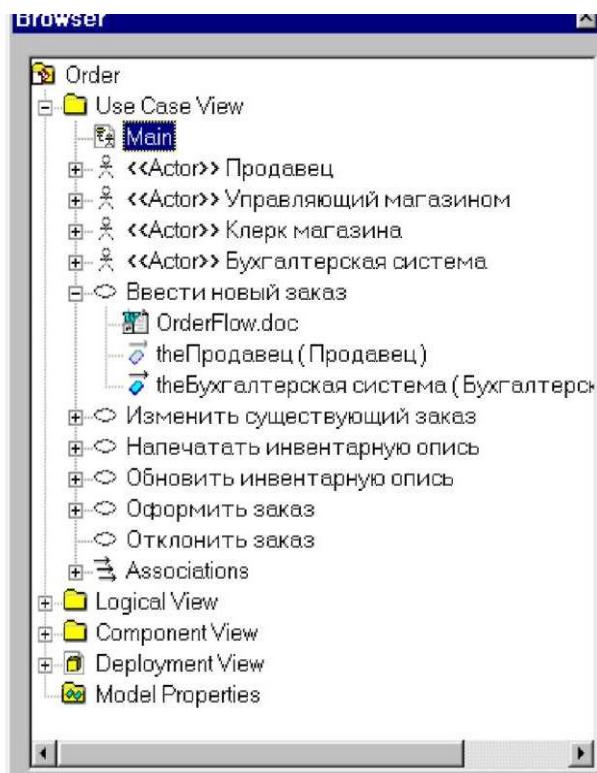


Рис. 2.3. Представление вариантов использования.

Логическое представление

Логическое представление, показанное на рис. 2.4, концентрируется на том, как система будет реализовывать поведение, описанное в вариантах использования. Оно дает подробную картину составных частей системы и описывает взаимодействие этих частей. Логическое представление включает, помимо прочего, конкретные требуемые классы, диаграммы классов и диаграммы состояний. С их помощью конструируется детальный проект создаваемой системы.

Логическое представление содержит:

- Классы.
- Диаграммы классов. Как правило, для описания системы используется несколько диаграмм классов, каждая из которых отображает некоторое подмножество всех классов системы.
- Диаграммы взаимодействия, применяемые для отображения объектов, участвующих в одном потоке событий варианта использования.
- Диаграммы состояний.
- Пакеты, являющиеся группами взаимосвязанных классов.

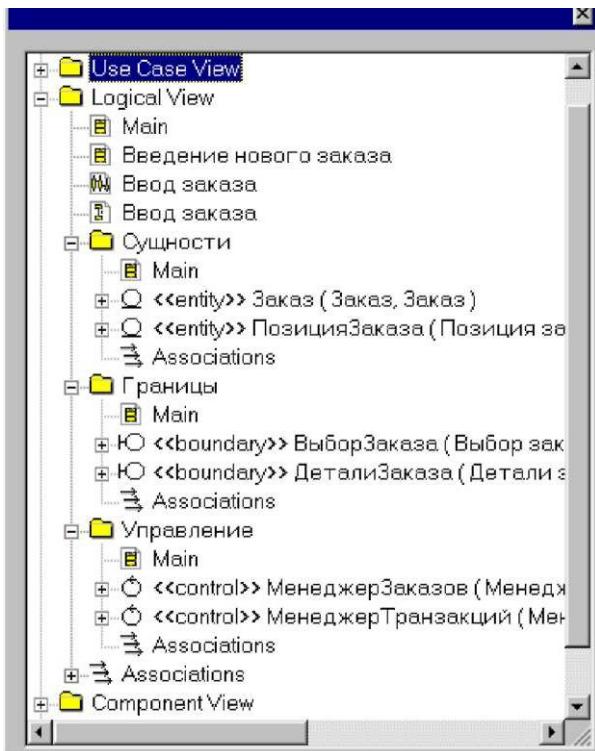


Рис. 2.4. Логическое представление системы.

Представление компонентов

Представление компонентов содержит:

- Компоненты, являющиеся физическими модулями кода.
- Диаграммы компонентов.
- Пакеты, являющиеся группами связанных компонентов.

Представление размещения

Последнее представление Rose - это представление размещения. Оно соответствует физическому размещению системы, которое может отличаться от ее логической архитектуры.

В представление размещения входят:

- Процессы, являющиеся потоками (threads), исполняемыми в отведенной для них области памяти.
- Процессоры, включающие любые компьютеры, способные обрабатывать данные. Любой процесс выполняется на одном или нескольких процессорах.
- Устройства, то есть любая аппаратура, не способная обрабатывать данные. К числу таких устройств относятся, например, терминалы ввода-вывода и принтеры.
- Диаграмма размещения.

2.2.3. Параметры настройки отображения

Изображение атрибутов и операций на диаграммах классов

В Rose имеется возможность настроить диаграммы классов так, чтобы:

- показывать все атрибуты и операции;
- скрыть операции;
- скрыть атрибуты;
- показывать только некоторые атрибуты или операции;
- показывать операции вместе с их полными сигнатурами или только их имена;
- показывать или не показывать видимость атрибутов и операций;
- показывать или не показывать стереотипы атрибутов и операций. Значения каждого параметра по умолчанию можно задать с помощью

окна, открываемого при выборе пункта меню Tools > Options. У данного класса на диаграмме можно:

- показать все атрибуты;
- скрыть все атрибуты;
- показать только выбранные вами атрибуты;
- подавить вывод атрибутов.

Подавление вывода атрибутов приведет не только к исчезновению атрибутов с диаграммы, но и к удалению линии, показывающей место расположения атрибутов в классе.

Существует два способа изменения параметров представления атрибутов на диаграмме. Можно установить нужные значения у каждого класса индивидуально. Можно также изменить значения нужных параметров по умолчанию до начала создания диаграммы классов. Внесенные таким образом изменения повлияют только на вновь создаваемые диаграммы.

Чтобы показать все атрибуты класса:

- 1 . Выделите на диаграмме нужный класс.
2. Щелкните на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выберите Options > Show All Attributes.

Чтобы показать у класса только избранные атрибуты:

- 1 . Выделите на диаграмме нужный вам класс.
2. Щелкните на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выберите Options > Select Compartment Items.
4. Укажите нужные вам атрибуты в окне Edit Compartment.

Чтобы подавить вывод всех атрибутов класса диаграммы:

1. Выделите на диаграмме нужный вам класс.
2. Щелкните на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выберите Options > Suppress Attributes.

Чтобы изменить принятый по умолчанию вид атрибута:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Для установки значений параметров отображения атрибутов по умолчанию воспользуйтесь контрольными переключателями Suppress Attributes и Show All Attributes. Изменение этих значений по умолчанию повлияет только на новые диаграммы. Вид существующих диаграмм классов не изменится.

Как и в случае атрибутов, имеется несколько вариантов представления операций на диаграммах.

- Показать все операции;
- Показать только некоторые операции.
- Скрыть все операции.
- Подавить вывод операций.

Кроме того, можно:

- Показать только имя операции. Это означает, что на диаграмме будет представлено только имя операции, но не аргументы или тип возвращаемого значения.
- Показать полную сигнатуру операции. На диаграмме будет представлено не только имя операции, но и все ее параметры, типы данных параметров и тип возвращаемого значения операции.

Чтобы показать все операции класса:

- 1 . Выделите на диаграмме нужный вам класс.
2. Щелкните на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выберите Options > Show All Operations.

Чтобы показать только выбранные операции класса:

- 1 . Выделите на диаграмме нужный вам класс.
2. Щелкните на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выберите Options > Select Compartment Items.
4. Укажите нужные вам операции в окне Edit Compartment.

Чтобы подавить вывод всех операций класса диаграммы:

- 1 . Выделите на диаграмме нужный вам класс.
2. Щелкните на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выберите Options > Suppress Operations.

Чтобы показать на диаграмме классов сигнатуру операции:

- 1 . Выделите на диаграмме нужный вам класс.
2. Щелкните на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выберите Options > Show Operation Signature.

Чтобы изменить принятый по умолчанию вид операции:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Для установки значений параметров отображения операций по умолчанию воспользуйтесь контрольными переключателями Suppress Operations, Show All Operations и Show Operation Signatures.

Чтобы показать видимость атрибута или операции класса:

- 1 . Выделите на диаграмме нужный вам класс.
2. Щелкните на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выберите Options > Show Visibility.

Чтобы изменить принятое по умолчанию значение параметра показа видимости:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Для установки параметров отображения видимости по умолчанию воспользуйтесь контрольным переключателем Show Visibility.

Для переключения между нотациями видимости Rose и UML:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Notation.
3. Для переключения между нотациями воспользуйтесь переключателем Visibility as Icons. Если этот переключатель помечен, будет использоваться нотация Rose. Если нет, то нотация UML. Изменение этого параметра повлияет только на новые диаграммы. Существующие диаграммы классов останутся прежними.

Глава 3. Выполнение учебного проекта

3.1. Система регистрации для ВУЗа. Постановка задачи

Перед руководителем информационной службы университета ставится задача разработки новой клиент-серверной системы регистрации студентов взамен старой системы на мейнфрейме. Новая система должна позволять студентам регистрироваться на курсы и просматривать свои табели успеваемости с персональных компьютеров, подключенных к локальной сети университета. Профессора должны иметь доступ к онлайновой системе, чтобы указать курсы, которые они будут читать, и проставить оценки за курсы.

Из-за недостатка средств университет не в состоянии заменить сразу всю существующую систему. Остается функционировать в прежнем виде база данных, содержащая всю информацию о курсах (каталог курсов). Эта база данных поддерживается реляционной СУБД. Новая система будет работать с существующей БД в режиме доступа, без обновления.

В начале каждого семестра студенты могут запросить каталог курсов, содержащий список курсов, предлагаемых в данном семестре. Информация о каждом курсе должна включать имя профессора, наименование кафедры и требования к предварительному уровню подготовки (прослушанным курсам).

Новая система должна позволять студентам выбирать 4 курса в предстоящем семестре. Дополнительно каждый студент может указать 2 альтернативных курса на тот случай, если какой-либо из выбранных им курсов окажется уже заполненным или отмененным. На каждый курс может записаться не более 10 и не менее 3 студентов (если менее 3, то курс будет отменен). В каждом семестре существует период времени, когда студенты могут изменить свои планы. В это время студенты должны иметь доступ к системе, чтобы добавить или удалить выбранные курсы. После того, как процесс регистрации некоторого студента завершен, система регистрации направляет информацию в расчетную систему, чтобы студент мог внести плату за семестр. Если курс окажется заполненным в процессе регистрации, студент должен быть извещен об этом до окончательного формирования его личного учебного плана.

В конце семестра студенты должны иметь доступ к системе для просмотра своих электронных табелей успеваемости. Поскольку эта информация конфиденциальная, система должна обеспечивать ее защиту от несанкционированного доступа.

Профессора должны иметь доступ к онлайновой системе, чтобы указать курсы, которые они будут читать, и просмотреть список студентов, записавшихся на их курсы. Кроме этого, профессора должны иметь возможность проставить оценки за курсы.

3.2. Составление гLOSSария проекта

Глоссарий предназначен для описания терминологии предметной области. Он может быть использован как неформальный *словарь данных* системы.

Курс	Учебный курс, предлагаемый университетом
Конкретный курс (Course Offering)	Конкретное чтение данного курса в конкретном семестре (один и тот же курс может вестись в нескольких параллельных сессиях). Включает точные дни недели и время.
Каталог курсов	Полный каталог всех курсов, предлагаемых университетом.
Расчетная система	Система обработки информации об оплате за курсы.
Оценка	Оценка, полученная студентом за конкретный курс.
Профессор	Преподаватель университета.
Табель успеваемости (Report Card)	Все оценки за все курсы, полученные студентом в данном семестре.
Список курса (Roster)	Список всех студентов, записавшихся на конкретный курс.
Студент	Личность, проходящая обучение в университете.
Учебный график (Schedule)	Курсы, выбранные студентом в текущем семестре.

3.3. Описание дополнительных спецификаций

Назначение дополнительных спецификаций - определить требования к системе регистрации курсов, которые не отражены в модели вариантов использования. Вместе они образуют полный набор требований к системе.

Дополнительные спецификации определяют нефункциональные требования к системе, такие, как надежность, удобство использования, производительность, сопровождаемость, а также ряд функциональных требований, являющихся общими для нескольких вариантов использования.

Функциональные возможности

Система должна обеспечивать многопользовательский режим работы.

Если конкретный курс оказывается заполненным в то время, когда студент формирует свой учебный график, включающий данный курс, то система должна известить его об этом.

Удобство использования

Пользовательский интерфейс должен быть совместимым с Windows 95/98.

Надежность

Система должна быть в работоспособном состоянии 24 часа в день 7 дней в неделю, время простоя - не более 10%.

Производительность

Система должна поддерживать до 2000 одновременно работающих с центральной базой данных пользователей, и до 500 пользователей, одновременно работающих с локальными серверами.

Безопасность

Система не должна позволять студентам изменять любые учебные графики, кроме своих собственных, а также не должна позволять профессорам модифицировать конкретные курсы, выбранные другими профессорами.

Только профессора имеют право ставить студентам оценки.

Только регистратор может изменять любую информацию о студентах.

Проектные ограничения

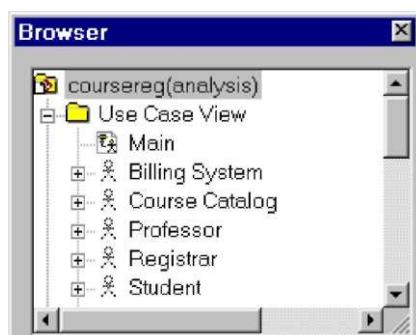
Система должна быть интегрирована с существующей системой каталога курсов, функционирующей на основе реляционной СУБД.

3.4. Создание модели вариантов использования

Действующие лица:

- Student (Студент) - записывается на курсы;
- Professor (Професор) - выбирает курсы для преподавания;
- Registrar (Регистратор) - формирует учебный план и каталог курсов, ведет все данные о курсах, профессорах и студентах;
- Billing System (Расчетная система) - получает от данной системы информацию по оплате за курсы;
- Course Catalog (Каталог курсов) - передает в систему информацию из каталога курсов, предлагаемых университетом.

Упражнение 1. Создание действующих лиц в среде Rational Rose



Чтобы поместить действующее лицо в браузер:

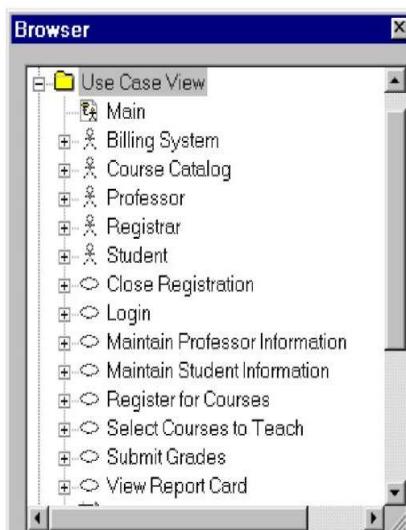
- 1 . Щелкните правой кнопкой мыши на пакете представления вариантов использования в браузере.
2. Выберите в открывшемся меню пункт New > Actor
3. В браузере появится новое действующее лицо под названием NewClass. Слева от его имени вы увидите пиктограмму действующего лица UML.
4. Выделив новое действующее лицо, введите его имя.
5. После создания действующих лиц сохраните модель под именем coursereg(analys) с помощью пункта меню File > Save.

Варианты использования:

Исходя из потребностей действующих лиц, выделяются следующие варианты использования:

- Login (Войти в систему);
- Register for Courses (Зарегистрироваться на курсы);
- View Report Card (Просмотреть табель успеваемости);
- Select Courses to Teach (Выбрать курсы для преподавания);
- Submit Grades (Проставить оценки);
- Maintain Professor Information (Вести информацию о профессорах);
- Maintain Student Information (Вести информацию о студентах);
- Close Registration (Закрыть регистрацию).

Упражнение 2. Создание вариантов использования в среде Rational Rose



Чтобы поместить вариант использования в браузер:

- 1 . Щелкните правой кнопкой мыши на пакете представления вариантов использования в браузере.
2. Выберите в появившемся меню пункт New > Use Case
3. Новый вариант использования под названием NewUseCase появится в браузере. Слева от него будет видна пиктограмма варианта использования UML.
4. Выделив новый вариант использования, введите его название.

Диаграмма вариантов использования:

Создайте диаграмму вариантов использования для системы регистрации. Требуемые для этого действия подробно перечислены далее. Готовая диаграмма вариантов использования должна выглядеть как на рис. 3.1.

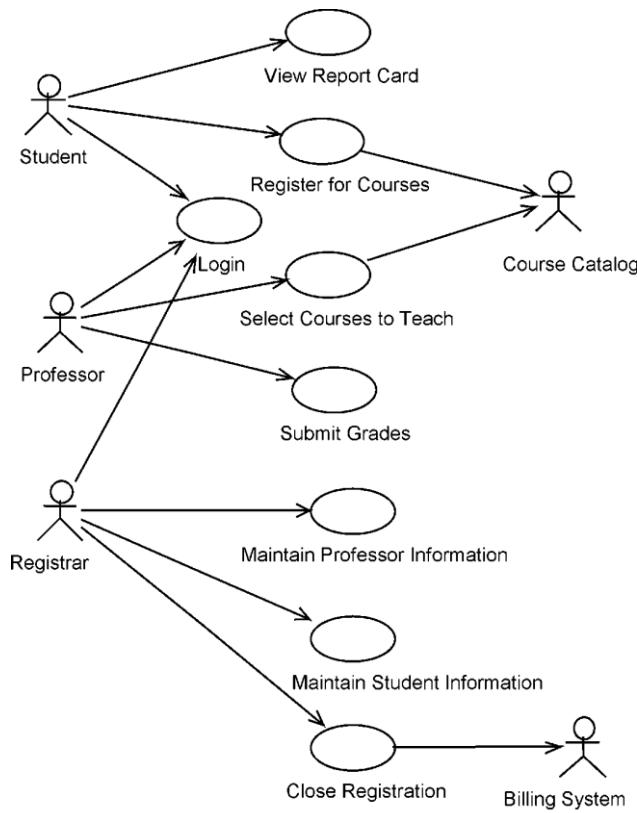


Рис. 3.1. Диаграмма вариантов использования для системы регистрации

В среде Rose диаграммы вариантов использования создаются в представлении вариантов использования. Главная диаграмма (Main) предлагается по умолчанию. Для моделирования системы можно затем разработать столько дополнительных диаграмм, сколько необходимо.

Чтобы получить доступ к главной диаграмме вариантов использования:

- 1 . Рядом с представлением вариантов использования в браузере щелкните на значке « + », это приведет к открытию данного представления.
2. Дважды щелкните на главной диаграмме Main, чтобы открыть её.

Строка заголовка изменится, включив фразу [Use Case Diagram:
Use Case view / Main].

Для создания новой диаграммы вариантов использования:

1. Щелкните правой кнопкой мыши на пакете представления вариантов использования в браузере.
2. Из всплывающего меню выберите пункт New > Use Case Diagram.
3. Выделив новую диаграмму, введите ее имя.
4. Дважды щелкните на названии этой диаграммы в браузере, чтобы открыть ее.

Упражнение 3. Построение диаграммы вариантов использования

- 1 . Откройте диаграмму вариантов использования Main.
2. Чтобы поместить действующее лицо или вариант использования на диаграмму, перетащите его мышью из браузера на диаграмму вариантов использования.
3. С помощью кнопки Unidirectional Association (Однонаправленная ассоциация) панели инструментов нарисуйте ассоциации между действующими лицами и вариантами использования.

Наличие общего варианта использования Login для трех действующих лиц позволяет обобщить их поведение и ввести новое действующее лицо Any User. Модифицированная диаграмма вариантов использования показана на рис. 3.2.

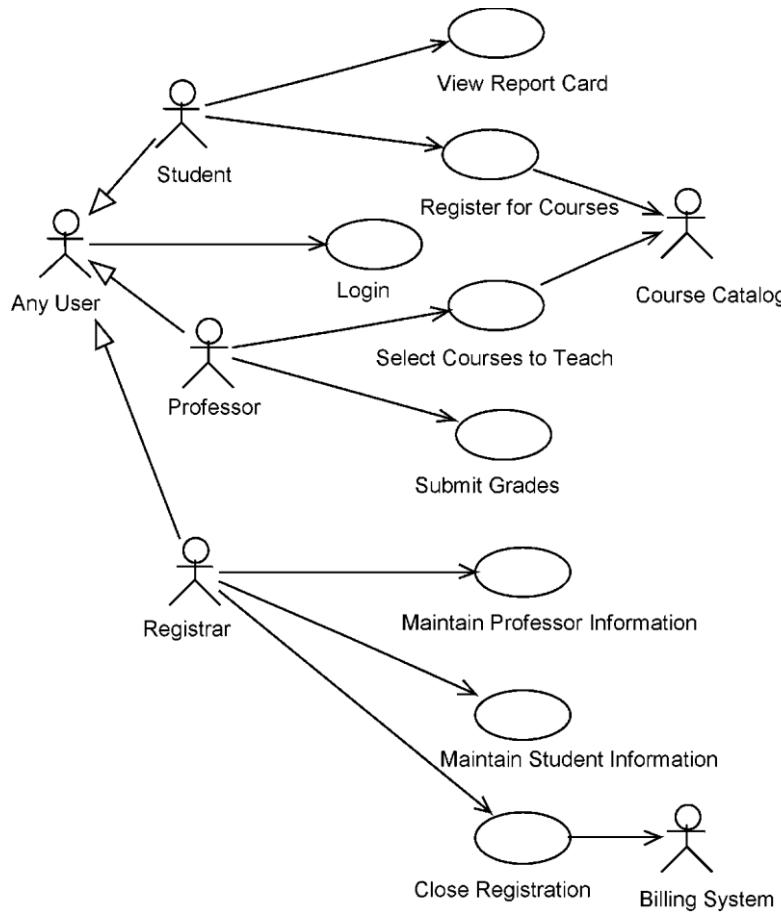


Рис. 3.2. Модифицированная диаграмма вариантов использования

Упражнение 4. Добавление описаний к вариантам использования

- Выделите в браузере вариант использования «Register for Courses».
- В окне документации введите следующее описание к этому варианту использования: «This use case allows a student to register for courses in the current semester» (Этот вариант использования дает студенту возможность зарегистрироваться на курсы в текущем семестре).
- Создайте с помощью MS Word три текстовых файла с описаниями вариантов использования Login (Войти в систему), Register for Courses (Зарегистрироваться на курсы) и Close Registration (Закрыть регистрацию).

Вариант использования Login:

Краткое описание

Данный вариант использования описывает вход пользователя в систему регистрации курсов.

Основной поток событий

Данный вариант использования начинает выполняться, когда пользователь хочет войти в систему регистрации курсов.

1 . Система запрашивает имя пользователя и пароль.

2. Пользователь вводит имя и пароль.

3. Система проверяет имя и пароль, после чего открывается доступ в систему.

Альтернативные потоки

Неправильное имя/пароль

Если во время выполнения **Основного потока** обнаружится, что пользователь ввел неправильное имя и/или пароль, система выводит сообщение об ошибке. Пользователь может вернуться к началу **Основного потока** или отказаться от входа в систему, при этом выполнение варианта использования завершается.

Предусловия

Отсутствуют.

Постусловия

Если вариант использования выполнен успешно, пользователь входит в систему. В противном случае состояние системы не изменяется.

Вариант использования Register for Courses:

Краткое описание

Данный вариант использования позволяет студенту зарегистрироваться на конкретные курсы в текущем семестре. Студент может изменить свой выбор (обновить или удалить курсы), если изменение выполняется в установленное время в начале семестра. Система каталога курсов предоставляет список всех конкретных курсов текущего семестра.

Основной поток событий

Данный вариант использования начинает выполняться, когда студент хочет зарегистрироваться на конкретные курсы или изменить свой график курсов.

- 1 . Система запрашивает требуемое действие (создать график, обновить график, удалить график).
2. Когда студент указывает действие, выполняется один из подчиненных потоков (создать, обновить, удалить или принять график).

Создать график

1. Система выполняет поиск в каталоге курсов доступных конкретных курсов и выводит их список.
2. Студент выбирает из списка 4 основных курса и 2 альтернативных курса.
3. После выбора система создает график студента.
4. Выполняется подчиненный поток «Принять график».

Обновить график

- 1 . Система выводит текущий график студента.
2. Система выполняет поиск в каталоге курсов доступных конкретных курсов и выводит их список.
3. Студент может обновить свой выбор курсов, удаляя или добавляя конкретные курсы.
4. После выбора система обновляет график.
5. Выполняется подчиненный поток «Принять график».

Удалить график

- 1 . Система выводит текущий график студента.
2. Система запрашивает у студента подтверждения удаления графика.
3. Студент подтверждает удаление.
4. Система удаляет график. Если график включает конкретные курсы, на которые записался студент, он должен быть удален из списков этих курсов.

Принять график

Для каждого выбранного, но еще не «записанного» конкретного курса в графике система проверяет выполнение студентом предварительных требований (прохождение определенных курсов), факт открытия конкретного курса и отсутствие конфликтов графика. Затем система добавляет студента в список выбранного конкретного курса. Курс фиксируется в графике и график сохраняется в системе.

Альтернативные потоки

Сохранить график

В любой момент студент может вместо принятия графика сохранить его. В этом случае шаг «Принять график» заменяется на следующий:

1. «Незаписанные» конкретные курсы помечаются в графике как «выбранные».
2. График сохраняется в системе.

Не выполнены предварительные требования, курс заполнен или имеют место конфликты графика

Если во время выполнения подчиненного потока «Принять график» система обнаружит, что студент не выполнил необходимые предварительные требования, или выбранный им конкретный курс заполнен, или имеют место конфликты графика, то выдается сообщение об ошибке. Студент

может либо выбрать другой конкретный курс и продолжить выполнение варианта использования, либо сохранить график, либо отменить операцию, после чего основной поток начнется с начала.

График не найден

Если во время выполнения подчиненных потоков «Обновить график» или «Удалить график» система не может найти график студента, то выдается сообщение об ошибке. После того, как студент подтвердит это сообщение, основной поток начнется с начала.

Система каталога курсов недоступна

Если окажется, что невозможно установить связь с системой каталога курсов, то будет выдано сообщение об ошибке. После того, как студент подтвердит это сообщение, вариант использования завершится.

Регистрация на курсы закончена

Если в самом начале выполнения варианта использования окажется, что регистрация на текущий семестр закончена, будет выдано сообщение и вариант использования завершится.

Удаление отменено

Если во время выполнения подчиненного потока «Удалить график» студент решит не удалять его, удаление отменяется, и основной поток начнется с начала.

Предусловия

Перед началом выполнения данного варианта использования студент должен войти в систему.

Постусловия

Если вариант использования завершится успешно, график студента будет создан, обновлен или удален. В противном случае состояние системы не изменится.

Вариант использования Close Registration:

Краткое описание

Данный вариант использования позволяет регистратору закрывать процесс регистрации. Конкретные курсы, на которые не записалось достаточного количества студентов (менее трех), отменяются. В расчетную систему передается информация о каждом студенте по каждому конкретному курсу, чтобы студенты могли внести оплату за курсы.

Основной поток событий

Данный вариант использования начинает выполняться, когда регистратор запрашивает прекращение регистрации.

- 1 . Система проверяет состояние процесса регистрации. Если регистрация еще выполняется, выдается сообщение и вариант использования завершается.
2. Для каждого конкретного курса система проверяет, ведет ли его какой-либо профессор, и записалось ли на него не менее трех студентов. Если эти условия выполняются, система фиксирует конкретный курс в каждом графике, который включает данный курс.
3. Для каждого студенческого графика проверяется наличие в нем максимального количества основных курсов; если их недостаточно, система пытается дополнить альтернативными курсами из списка данного графика. Выбирается первый доступный альтернативный курс. Если таких курсов нет, то никакое дополнение не происходит.
4. Система закрывает все конкретные курсы. Если в каком-либо конкретном курсе оказывается менее трех студентов (с учетом добавлений, сделанных в п.3), система отменяет его и исключает из каждого содержащего его графика.
5. Система рассчитывает плату за обучение для каждого студента в текущем семестре и направляет информацию в расчетную систему. Расчетная система посылает студентам счета для оплаты с копией их окончательных графиков.

Альтернативные потоки

Конкретный курс никто не ведет

Если во время выполнения основного потока обнаруживается, что некоторый конкретный не ведется никаким профессором, то этот курс отменяется. Система исключает данный курс из каждого содержащего его графика.

Расчетная система недоступна

Если невозможно установить связь с расчетной системой, через некоторое установленное время система вновь попытается связаться с ней. Попытки будут повторяться до тех пор, пока связь не установится.

Предусловия

Перед началом выполнения данного варианта использования регистратор должен войти в систему.

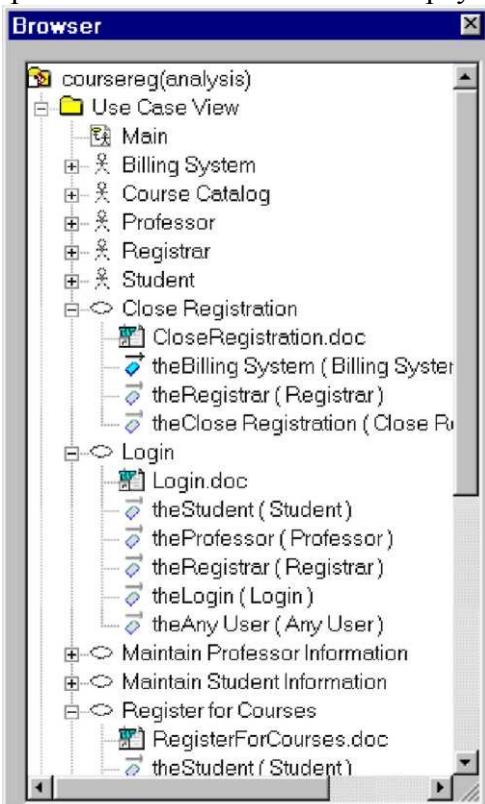
Постусловия

Если вариант использования завершится успешно, регистрация закрывается. В противном случае состояние системы не изменится.

Упражнение 5. Прикрепление файла к варианту использования

- 1 . Щелкните правой кнопкой мыши на варианте использования.
2. В открывшемся меню выберите пункт Open Specification
3. Перейдите на вкладку файлов.
4. Щелкните правой кнопкой мыши на белом поле и из открывшегося меню выберите пункт Insert File.
5. Укажите созданный ранее файл и нажмите на кнопку Open, чтобы прикрепить файл к варианту использования.

В результате представление вариантов использования в браузере примет следующий вид:



Удаление вариантов использования и действующих лиц

Существует два способа удалить элемент модели - из одной диаграммы или из всей модели. Чтобы удалить элемент модели из диаграммы:

- 1 . Выделите элемент на диаграмме.
2. Нажмите на клавишу Delete.
3. Обратите внимание, что, хотя элемент и удален с диаграммы, он остался в браузере и на других диаграммах системы.

Чтобы удалить элемент из модели:

1. Выделите элемент на диаграмме.
2. Выберите пункт меню Edit > Delete from Model или нажмите сочетание клавиш CTRL+D.

3.5. Анализ системы

3.5.1. Архитектурный анализ

Принятие соглашений по моделированию Включает:

- Используемые диаграммы и элементы модели;
- Правила их применения;
- Соглашения по именованию элементов;
- Организация модели (пакеты).

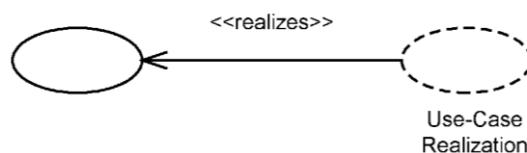
Пример соглашений моделирования:

- Имена вариантов использования должны быть короткими глагольными фразами.
- Для каждого варианта использования должен быть создан пакет Use-Case Realization, включающий:
 - по крайней мере одну реализацию варианта использования;
 - диаграмму «View Of Participating Classes» (VOPC).
- Имена классов должны быть существительными, соответствующими, по возможности, понятиям предметной области.
- Имена классов должны начинаться с заглавной буквы.
- Имена атрибутов и операций должны начинаться со строчной буквы.
- Составные имена должны быть сплошными, без подчеркиваний, каждое отдельное слово должно начинаться с заглавной буквы.

Реализация варианта использования (Use-Case Realization):

Описывает реализацию конкретного варианта использования в терминах взаимодействующих объектов и представляется с помощью набора диаграмм (диаграмм классов, реализующих вариант использования, и диаграмм взаимодействия (диаграмм последовательности и кооперативных диаграмм), отражающих взаимодействие объектов в процессе реализации варианта использования).

Рис. 3.3. Реализация варианта использования



Идентификация ключевых абстракций

Заключается в предварительном определении классов системы (классов анализа). Источники - знание предметной области, требования к системе, глоссарий. Классы анализа для системы регистрации показаны на рис. 3.4:

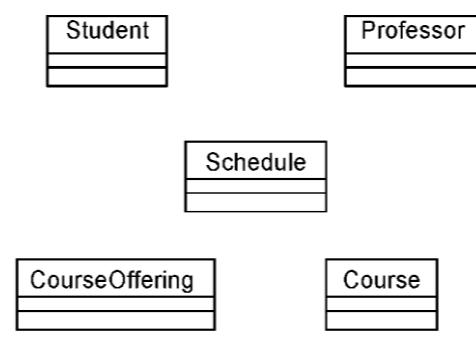
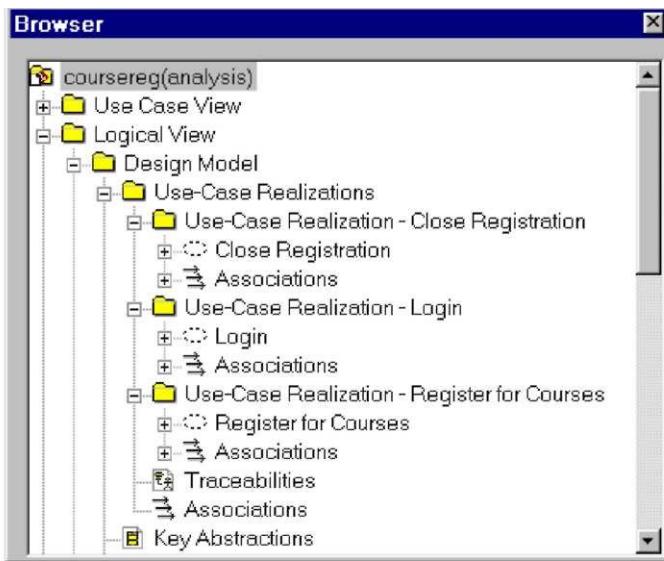


Рисунок 3.4. Классы анализа системы регистрации

Упражнение 6. Создание структуры модели и классов анализа в соответствии с требованиями архитектурного анализа

Структура логического представления браузера должна иметь следующий вид:



Создание пакетов и диаграммы Traceabilities:

- 1 . Щелкните правой кнопкой мыши на логическом представлении браузера.
2. В открывшемся меню выберите пункт New > Package
3. Назовите новый пакет Design Model.
4. Создайте аналогичным образом пакеты Use-Case Realizations, Use-Case Realization - Close Registration, Use-Case Realization - Login и Use-Case Realization - Register for Courses.
5. В каждом из пакетов типа Use-Case Realization создайте соответствующие кооперации Close Registration, Login и Register for Courses (каждая кооперация представляет собой вариант использования со стереотипом «use-case realization», который задается в спецификации варианта использования).

Создайте в пакете Use-Case Realizations новую диаграмму вариантов использования с названием Traceabilities и постройте ее в соответствии с рис. 3.5.

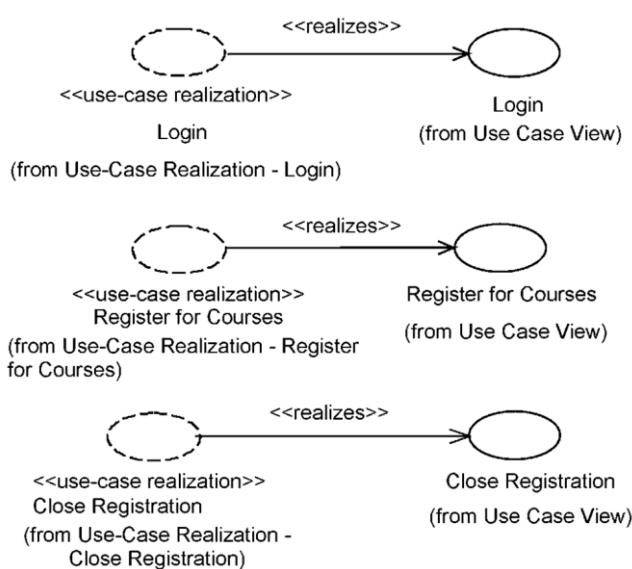


Рис. 3.5. Диаграмма Traceabilities

Создание классов анализа и соответствующей диаграммы Key Abstractions:

- 1 . Щелкните правой кнопкой мыши на пакете Design Model.
2. Выберите в открывшемся меню пункт New > Class. Новый класс под названием NewClass появится в браузере.
3. Выделите его и введите имя Student.
4. Создайте аналогичным образом классы Professor, Schedule, Course и CourseOffering.
5. Щелкните правой кнопкой мыши на пакете Design Model.

6. В открывшемся меню выберите пункт New > Class Diagram.
7. Назовите новую диаграмму классов Key Abstractions.
8. Чтобы расположить вновь созданные классы на диаграмме классов, откройте ее и перетащите классы на открытую диаграмму мышью. Диаграмма классов должна выглядеть как на рис. 3.4.

3.5.2. Анализ вариантов использования

Идентификация классов, участвующих в реализации потоков событий варианта использования

В потоках событий варианта использования выявляются классы трех типов:

1. **Граничные классы (Boundary)** - служат посредниками при взаимодействии внешних объектов с системой. Как правило, для каждой пары «действующее лицо - вариант использования» определяется один граничный класс. Типы граничных классов: пользовательский интерфейс (обмен информацией с пользователем, без деталей интерфейса - кнопок, списков, окон), системный интерфейс и аппаратный интерфейс (используемые протоколы, без деталей их реализации).
2. **Классы-сущности (Entity)** - представляют собой ключевые абстракции (понятия) разрабатываемой системы. Источники выявления классов-сущностей: ключевые абстракции, созданные в процессе архитектурного анализа, глоссарий, описание потоков событий вариантов использования.
3. **Управляющие классы (Control)** - обеспечивают координацию поведения объектов в системе. Могут отсутствовать в некоторых вариантах использования, ограничивающихся простыми манипуляциями с хранимыми данными. Как правило, для каждого варианта использования определяется один управляющий класс. Примеры управляющих классов: менеджер транзакций, координатор ресурсов, обработчик ошибок.

Пример набора классов, участвующих в реализации варианта использования Register for Courses, приведен на рис. 3.6.

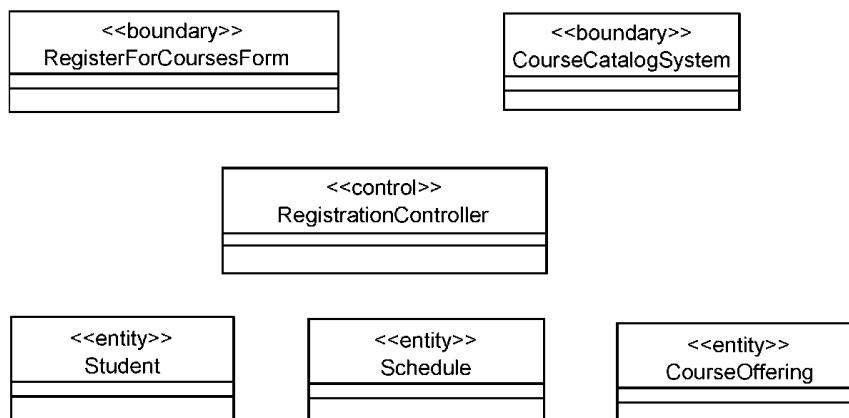


Рис. 3.6. Классы, участвующие в реализации варианта использования Register for Courses

Упражнение 7. Создание классов, участвующих в реализации варианта использования Register for Courses, и диаграммы классов «View Of Participating Classes» (VOPC)

1. Щелкните правой кнопкой мыши на пакете Design Model.
2. Выберите в открывшемся меню пункт New > Class. Новый класс под названием NewClass появится в браузере.
3. Выделите его и введите имя RegisterForCoursesForm.
4. Щелкните правой кнопкой мыши на классе RegisterForCoursesForm.
5. В открывшемся меню выберите пункт Open Specification.
6. В поле стереотипа выберите Boundary и нажмите на кнопку OK.

7. Создайте аналогичным образом классы CourseCatalogSystem со стереотипом Boundary и RegistrationController со стереотипом Control.
8. Назначьте классам Schedule, CourseOffering и Student стереотип Entity.
9. Щелкните правой кнопкой мыши на кооперации Register for Courses в пакете Use-Case Realization - Register for Courses.
 10. В открывшемся меню выберите пункт New > Class Diagram.
 11. Назовите новую диаграмму классов VOPC (classes only).
 12. Откройте ее и перетащите классы на открытую диаграмму в соответствии с рис. 3.6.

Распределение поведения, реализуемого вариантом использования, между классами

Реализуется с помощью диаграмм взаимодействия (диаграмм последовательности и кооперативных диаграмм). В первую очередь строится диаграмма (одна или более), описывающая основной поток событий и его подчиненные потоки. Для каждого альтернативного потока событий строится отдельная диаграмма. Примеры:

- обработка ошибок;
- контроль времени выполнения;
- обработка неправильных вводимых данных.

Неследесообразно описывать тривиальные потоки событий (например, в потоке участвует только один объект).

Упражнение 8. Создание диаграмм взаимодействия

Создадим диаграммы последовательности и кооперативные диаграммы для основного потока событий варианта использования Register for Courses. Готовые диаграммы последовательности должны иметь вид, как на рис. 3.7 - 3.11.

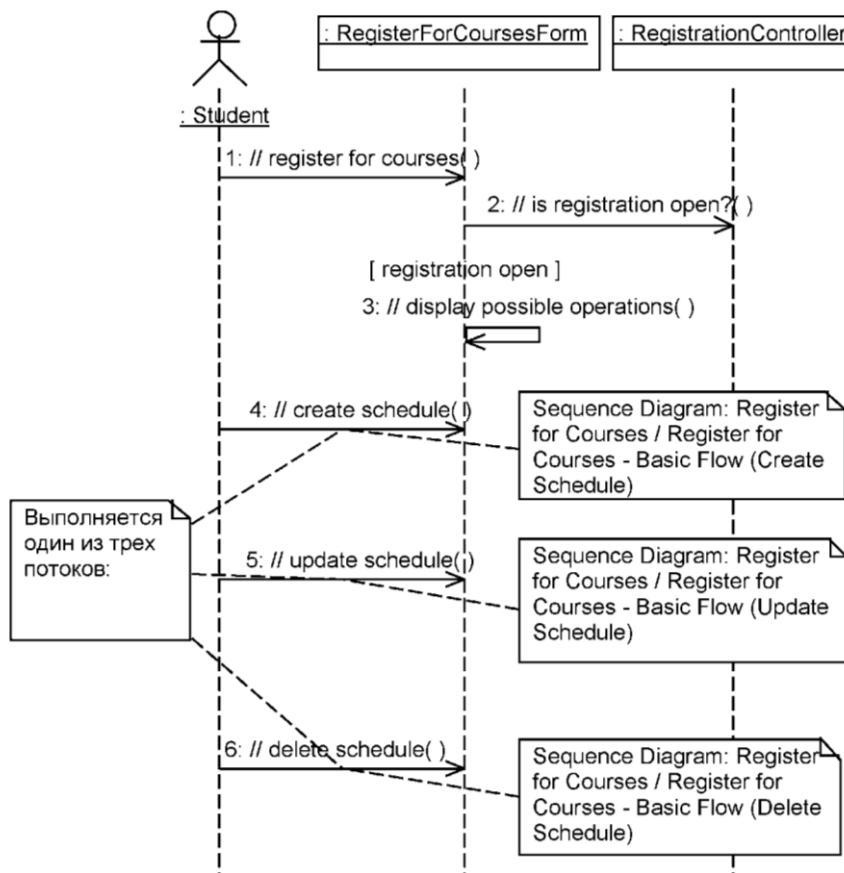


Рис. 3.7. Диаграмма последовательности Register for Courses - Basic Flow

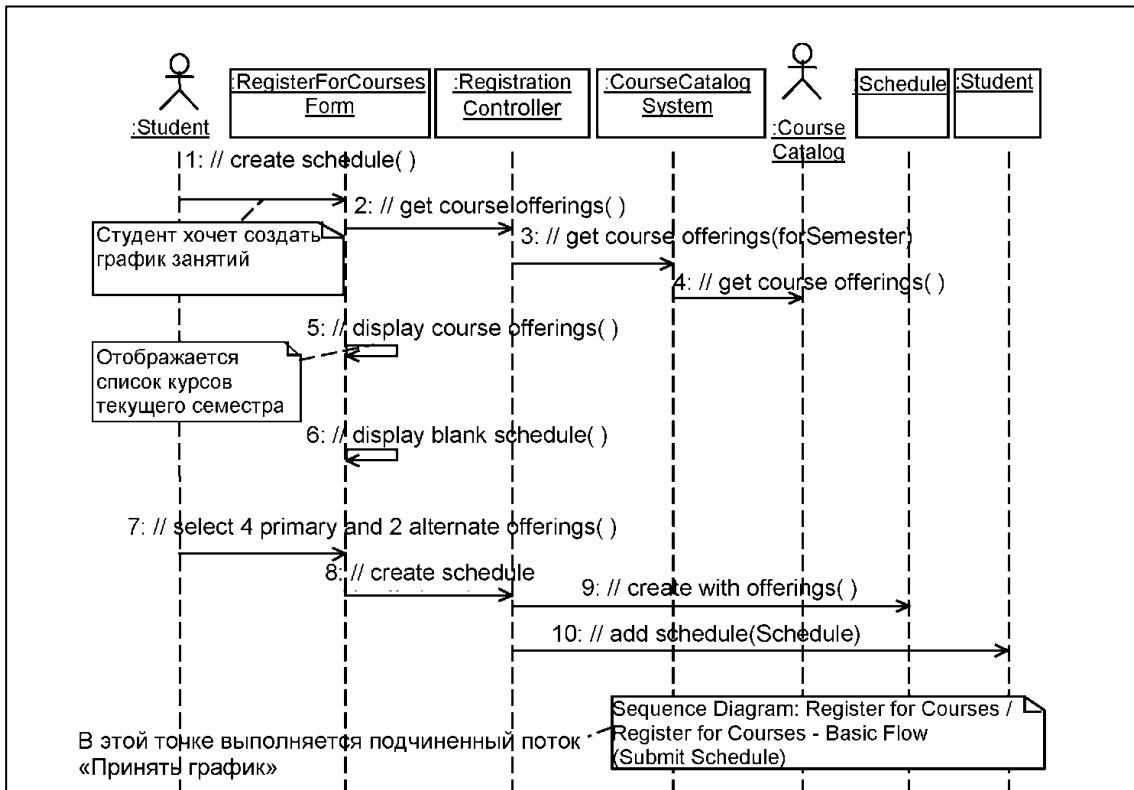


Рис. 3.8. Диаграмма последовательности Register for Courses - Basic Flow (Create Schedule)

Настройка

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку диаграмм.
3. Контрольные переключатели Sequence Numbering, Collaboration Numbering должны быть помечены, а Focus of Control - нет.
4. Нажмите OK, чтобы выйти из окна параметров.

Создание диаграммы последовательности

- 1 . Щелкните правой кнопкой мыши на кооперации Register for Courses в пакете Use-Case Realization - Register for Courses.
2. В открывшемся меню выберите пункт New > Sequence Diagram.
3. Назовите новую диаграмму Register for Courses - Basic Flow.
4. Дважды щелкните на ней, чтобы открыть ее

Добавление на диаграмму действующего лица, объектов и сообщений

- 1 . Перетащите действующее лицо Student из браузера на диаграмму.
2. Перетащите классы RegisterForCoursesForm и RegistrationController из браузера на диаграмму.
3. На панели инструментов нажмите кнопку Object Message (Сообщение объекта).
4. Проведите мышью от линии жизни действующего лица Student к линии жизни объекта RegisterForCoursesForm.
5. Выделив сообщение, введите его имя: // register for courses.
6. Повторите действия 3-5, чтобы поместить на диаграмму остальные сообщения, как показано на рис. 3.7 (для рефлексивного сообщения 3 используется кнопка Message to Self).

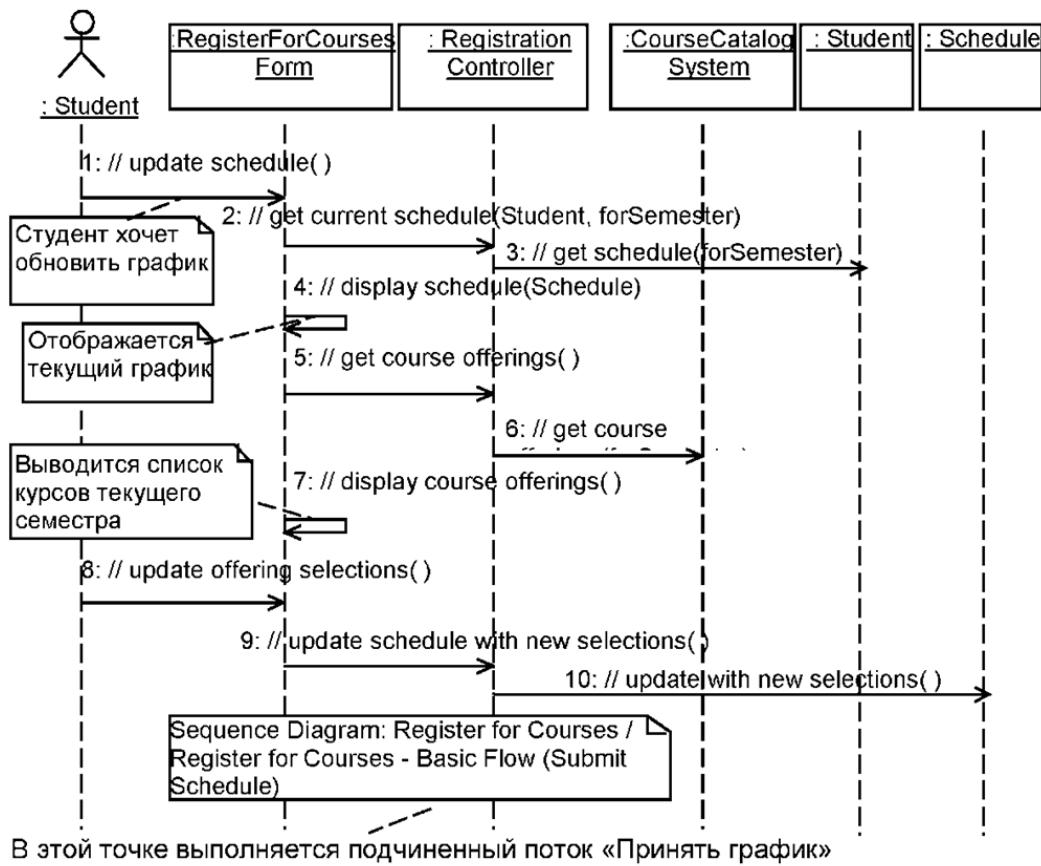


Рис.3.9. Диаграмма последовательности Register for Courses – Basic Flow (Update Schedule).

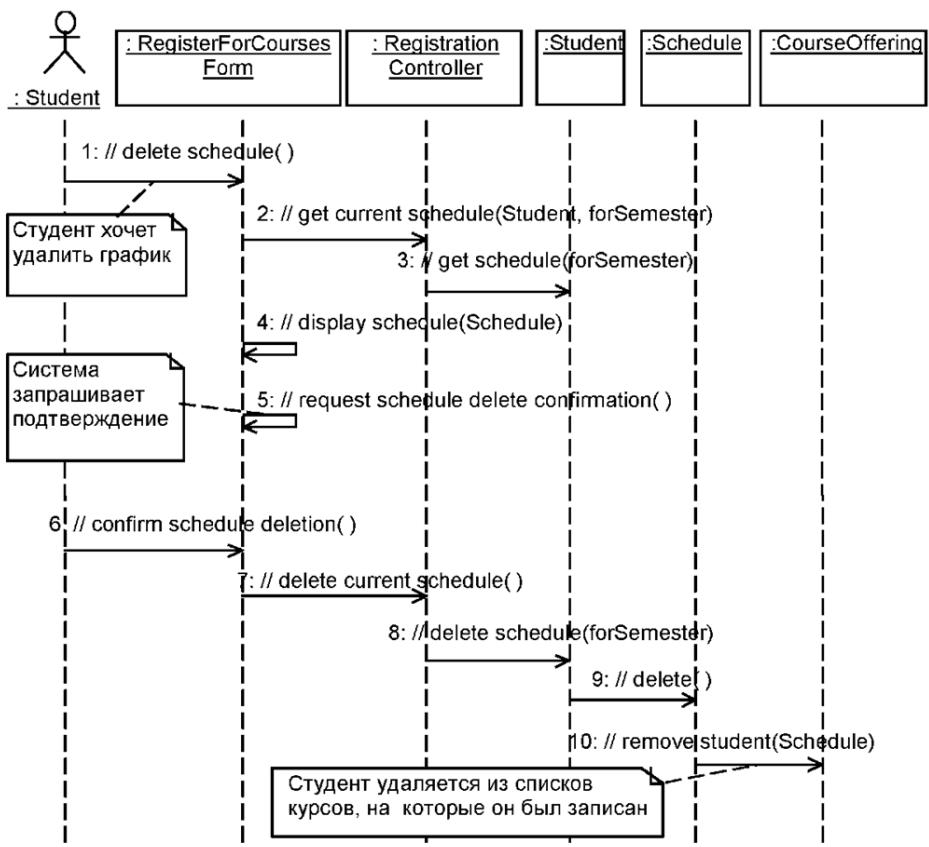


Рис. 3.10. Диаграмма последовательности Register for Courses - Basic Flow (Delete Schedule)

Соотнесение сообщений с операциями

- Щелкните правой кнопкой на сообщении 1, // register for courses.
- В открывшемся меню выберите пункт <new operation>. Появится окно спецификации операции.
- В поле имени оставьте имя сообщения - // register for courses.
- Нажмите на кнопку OK, чтобы закрыть окно спецификации операции и вернуться на диаграмму.
- Повторите действия 1 - 4, пока не соотнесете с операциями все остальные сообщения.

Выполните аналогичные действия для создания диаграмм последовательности, показанных на рис. 3.8 - 3.11. Обратите внимание, что на диаграмме рис. 3.11 появился объект нового класса PrimaryScheduleOfferingInfo (класса ассоциаций, описывающего связь между классами Schedule и OfferingInfo), который нужно предварительно создать.

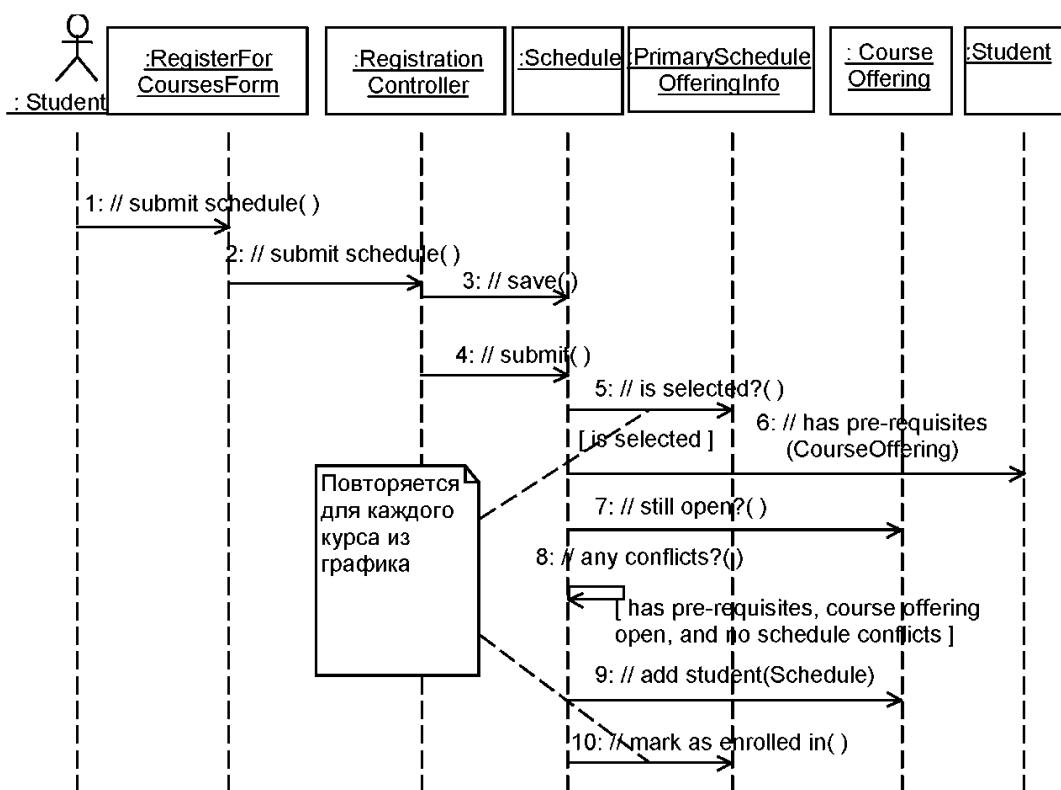


Рис. 3.11. Диаграмма последовательности Register for Courses - Basic Flow (Submit Schedule)

Создание примечаний

Чтобы поместить на диаграмму примечание:

- Нажмите на панели инструментов кнопку Note.
- Щелкните мышью в том месте диаграммы, куда собираетесь поместить примечание.
- Выделив новое примечание, введите туда текст.
- Чтобы прикрепить примечание к элементу диаграммы, на панели инструментов нажмите кнопку Anchor Notes To Item (Прикрепить примечания к элементу).
- Нажав левую кнопку мыши, проведите указатель от примечания до элемента диаграммы, с которым оно будет связано. Между примечанием и элементом возникнет штриховая линия.
- Чтобы создать примечание-ссылку на другую диаграмму (как это сделано на диаграмме рис. 3.7 и других), создайте пустое примечание (без текста) и перетащите на него из браузера нужную диаграмму.

Кроме примечаний, на диаграмму можно поместить также и текстовую область. С ее помощью можно, например, добавить к диаграмме заголовок.

Чтобы поместить на диаграмму текстовую область:

1. На панели управления нажмите кнопку Text Box.
2. Щелкните мышью внутри диаграммы, чтобы поместить туда текстовую область.
3. Выделив эту область, введите в неё текст.

Создание кооперативной диаграммы

Для создания кооперативной диаграммы достаточно открыть диаграмму последовательности и нажать клавишу F5.

Определение обязанностей (responsibilities), атрибутов и ассоциаций классов

Обязанность (responsibility) - действие, которое объект обязан выполнять по запросу других объектов. Обязанность преобразуется в одну или более операций класса на шаге проектирования. Обязанности определяются, исходя из сообщений на диаграммах взаимодействия, и документируются в классах в виде операций «анализа», которые появляются там автоматически в процессе построения диаграмм взаимодействия (соотнесения сообщений с операциями).

Так, диаграмма классов VOPC (classes only) (рис. 3.6) после построения диаграмм взаимодействия в упражнении 8 должна принять вид, изображенный на рис. 3.12.

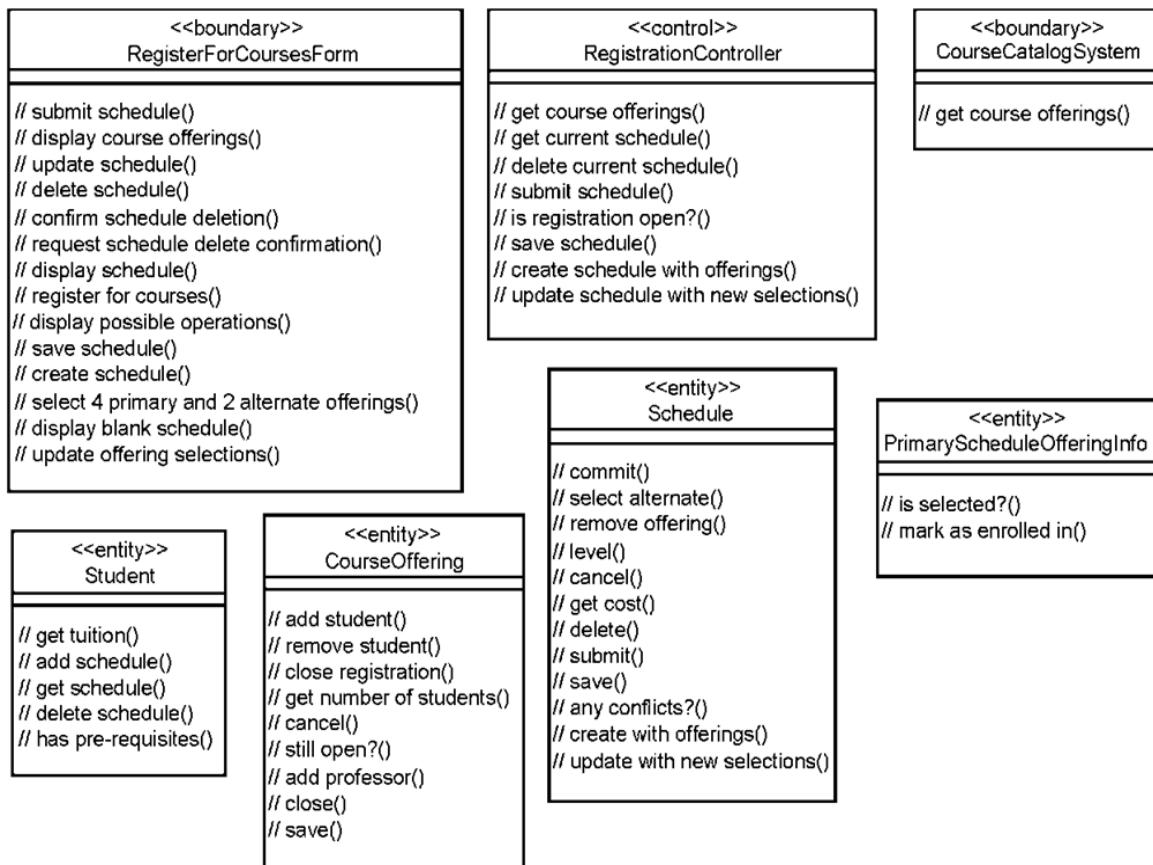


Рис. 3.12. Диаграмма классов VOPC (classes only) с операциями «анализа»

Атрибуты классов анализа определяются, исходя из знаний о предметной области, требований к системе и глоссария.

Упражнение 9. Добавление атрибутов к классам Настройка

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Diagram.
3. Убедитесь, что переключатель Show All Attributes помечен.
4. Убедитесь, что переключатели Suppress Attributes и Suppress Operations не помечены.

Добавление атрибутов

1. Щелкните правой кнопкой мыши на классе Student.

2. В открывшемся меню выберите пункт New Attribute.
3. Введите новый атрибут address
4. Нажмите клавишу Enter.
5. Повторите шаги 1-4, добавив атрибуты name и studentID.
6. Добавьте атрибуты к классам CourseOffering, Schedule PrimaryScheduleOfferingInfo, как показано на рис. 3.13.

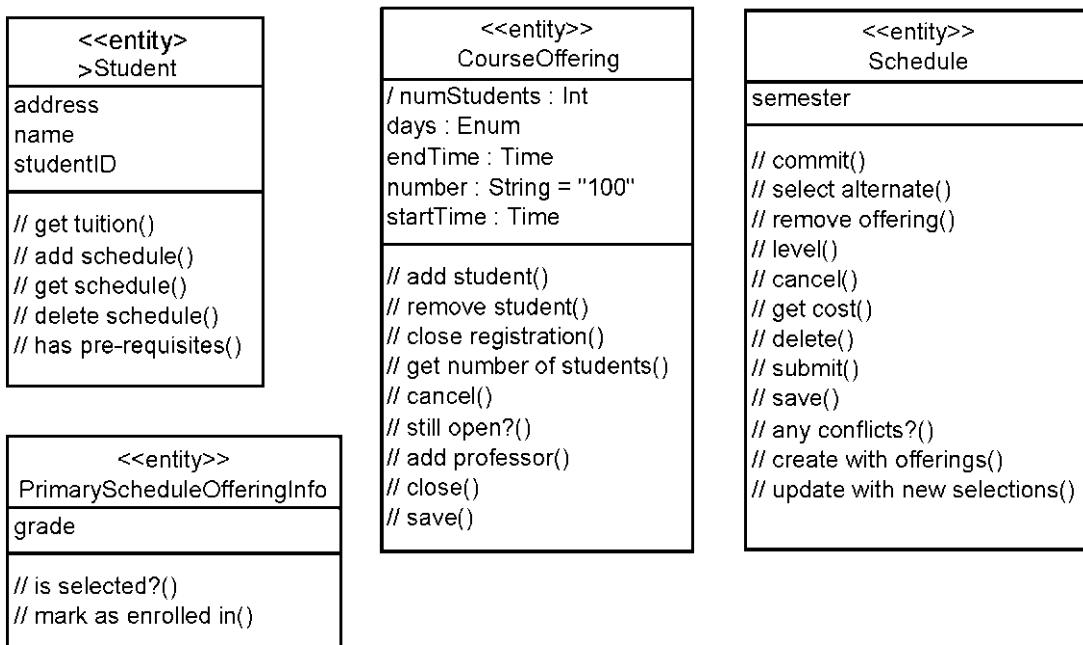


Рис. 3.13. Классы с операциями «анализа» и атрибутами

Связи между классами (ассоциации) определяются на основе диаграмм взаимодействия. Если два объекта взаимодействуют (обмениваются сообщениями), между ними должна существовать связь (путь взаимодействия). Для ассоциаций задаются множественность и, возможно, направление навигации. Могут использоваться множественные ассоциации, агрегации и классы ассоциаций.

Упражнение 10. Добавление связей

Добавим связи к классам, принимающим участие в варианте использования Register for Courses. Для отображения связей между классами построим три новых диаграммы классов в кооперации Register for Courses пакета Use-Case Realization - Register for Courses (рис. 3.14 - 3.16).

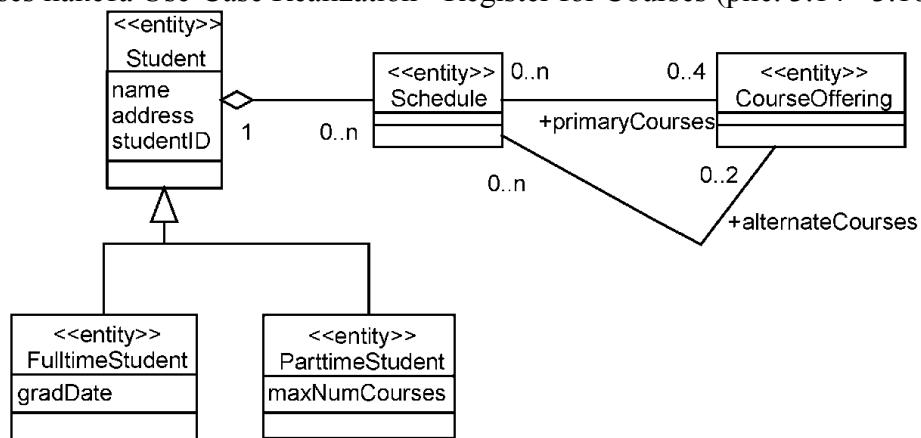


Рис. 3.14. Диаграмма Entity Classes (классы-сущности)

Добавлены два новых класса - подклассы **FulltimeStudent** (студент очного отделения) и **ParttimeStudent** (студент вечернего отделения).

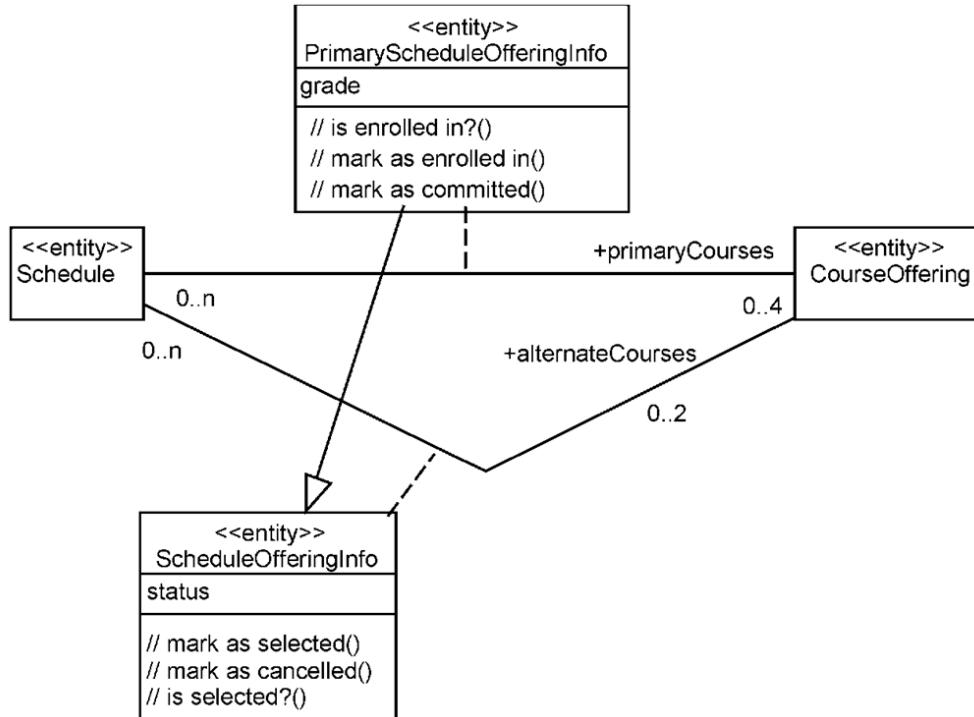


Рис. 3.15. Диаграмма CourseOfferingInfo

Чтобы создать рефлексивную ассоциацию:

1. На панели инструментов диаграммы нажмите кнопку **Association**.
2. Проведите линию ассоциации от класса до какого-нибудь места вне класса.
3. Отпустите кнопку мыши.
4. Проведите линию ассоциации назад к классу.

Создание агрегаций

1 . Нажмите кнопку **Aggregation** панели инструментов.

2. Проведите линию агрегации от класса-части к целому.

Чтобы поместить на диаграмму классов рефлексивную агрегацию:

- 1 . На панели инструментов диаграммы нажмите кнопку **Aggregation**.
2. Проведите линию агрегации от класса до какого-нибудь места вне класса.
3. Отпустите кнопку мыши.
4. Проведите линию агрегации назад к классу.

Создание обобщений

При создании обобщения может потребоваться перенести некоторые атрибуты или операции из одного класса в другой. Если, например, понадобится перенести их из подкласса в суперкласс `Employee`, в браузере для этого достаточно просто перетащить атрибуты или операции из одного класса в другой. Не забудьте удалить другую копию атрибута из второго подкласса, если он имеется.

Чтобы поместить обобщение на диаграмму классов:

1. Нажмите кнопку **Generalization** панели инструментов.
2. Проведите линию обобщения от подкласса к суперклассу.

Спецификации связей

Спецификации связей касаются имен ассоциаций, ролевых имен, множественности и классов ассоциаций.

Чтобы задать множественность связи:

1. Щелкните правой кнопкой мыши на одном конце связи.
2. В открывшемся меню выберите пункт **Multiplicity**.
3. Укажите нужную множественность.
4. Повторите то же самое для другого конца связи.

На данной диаграмме показаны классы ассоциаций, описывающие связи между классами Schedule и CourseOffering и добавлен суперкласс ScheduleOfferingInfo. Данные и операции, содержащиеся в этом классе (status - курс включен в график или отменен), относятся как к основным, так и к альтернативным курсам, в то время как оценка (grade) и окончательное включение курса в график могут иметь место только для основных курсов.

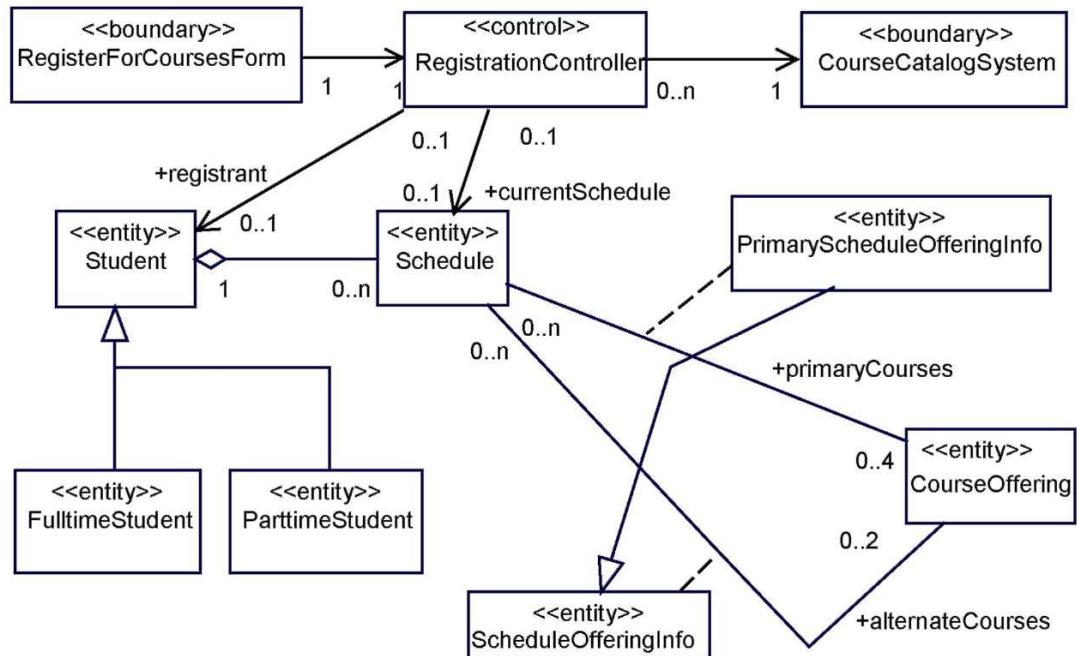


Рис. 3.16. Полная диаграмма классов VOPC (без атрибутов и операций)

Создание ассоциаций

Ассоциации создают непосредственно на диаграмме классов. Панель инструментов диаграммы классов содержит кнопки для создания как одно, так и двунаправленных ассоциаций. Чтобы на диаграмме классов создать ассоциацию:

1. Нажмите на панели инструментов кнопку Association.
2. Проведите мышью линию ассоциации от одного класса к другому.

Чтобы задать возможности навигации по ассоциации:

1. Щелкните правой кнопкой мыши на связи с того конца, на котором хотите показать стрелку.
2. В открывшемся меню выберите пункт Navigable.

Чтобы задать имя связи:

1. Выделите нужную связь.
2. Введите ее имя.

Чтобы задать связи ролевое имя:

1. Щелкните правой кнопкой мыши на ассоциации с нужного конца.
2. В открывшемся меню выберите пункт role Name.
3. Введите ролевое имя.

Чтобы задать элемент связи (класс ассоциаций):

1. Откройте окно спецификации требуемой связи.
2. Перейдите на вкладку Detail.
3. Задайте элемент связи в поле Link Element.

Задание для самостоятельной работы

Выполнить анализ варианта использования Close Registration и построить соответствующие диаграммы взаимодействия и классов.

3.6. Проектирование системы

3.6.1. Проектирование архитектуры

Цели проектирования архитектуры системы:

- анализ взаимодействий между классами анализа, выявление подсистем и интерфейсов;
- уточнение архитектуры с учетом возможностей повторного использования;
- идентификация архитектурных решений и механизмов, необходимых для проектирования системы.

Вводятся глобальные пакеты:

- базисные (foundation) классы (списки, очереди и т.д.);
- обработчики ошибок (error handling classes);
- математические библиотеки;
- утилиты;
- библиотеки других поставщиков. Определяются проектные классы (design classes):
- класс анализа отображается в проектный класс, если он простой или представляет единственную логическую абстракцию;
- сложный класс анализа может быть разбит на несколько классов, преобразован в пакет или в подсистему.

Примеры возможных подсистем:

- классы, обеспечивающие сложный комплекс услуг (например, обеспечение безопасности и защита);
- граничные классы, реализующие сложный пользовательский интерфейс или интерфейс с внешними системами;
- различные продукты: коммуникационное ПО (middleware, поддержка COM/CORBA), доступ к базам данных, типы и структуры данных (стеки, списки, очереди), общие утилиты (математические библиотеки), различные прикладные продукты.

Принятие решения о преобразовании класса в подсистему определяется опытом и знаниями архитектора проекта. Соглашения по проектированию интерфейсов:

- Имя интерфейса: короткое (одно-два слова), отражающее его роль в системе.
- Описание интерфейса: должно отражать его обязанности (размер - небольшой абзац).
- Описание операций: имя, отражающее результат операции, ключевые алгоритмы, возвращаемое значение, параметры с типами.
- Документирование интерфейса: характер использования операций и порядок их выполнения (показывается с помощью диаграмм последовательности), тестовые планы и сценарии и так далее. Вся эта информация объединяется в специальный пакет со стереотипом <<subsystem>>, который содержит элементы, образующие подсистему, диаграммы последовательности и/или кооперативные диаграммы, описывающие взаимодействие элементов при реализации операций интерфейса, и другие диаграммы.
- Класс <<subsystem proxy>> непосредственно реализует интерфейс и управляет реализацией его операций.
- Все интерфейсы должны быть полностью определены в процессе проектирования архитектуры, поскольку они будут служить в качестве точек синхронизации при параллельной разработке.

Выделение архитектурных уровней:

Application Layer - содержит элементы прикладного уровня (пользовательский интерфейс);

Business Services Layer - содержит элементы, реализующие бизнес-логику приложений (наиболее устойчивая часть системы);

Middleware Layer - обеспечивает сервисы, независимые от платформы.

Пример выделения архитектурных уровней и объединения элементов модели в пакеты для системы регистрации приведен на рис. 3.17.

Для того чтобы поместить класс в пакет, достаточно просто перетащить его в браузере на нужный пакет.

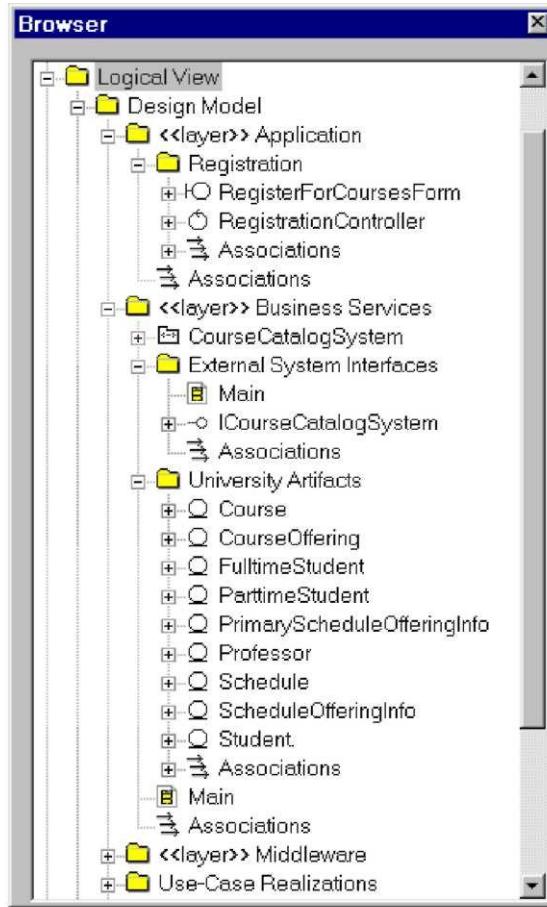


Рис. 3.17. Структура логического представления модели на шаге проектирования

Данное представление отражает следующие решения, принятые архитектором:

- Выделены три архитектурных уровня (созданы три пакета со стереотипом <<layer>>);
- В пакете Application создан пакет Registration, куда включены граничные и управляющие классы;
- Граничный класс CourseCatalogSystem преобразован в подсистему (пакет CourseCatalogSystem со стереотипом <<subsystem>>)
- В пакет Business Services, помимо подсистемы CourseCatalogSystem, включены еще два пакета: пакет External System Interfaces включает интерфейс с подсистемой CourseCatalogSystem (класс ICourseCatalogSystem со стереотипом <<Interface>>), а пакет University Artifacts - все классы-сущности.

Структура и диаграммы пакета (подсистемы) CourseCatalogSystem показана на рис. 3.18 - 3.22.

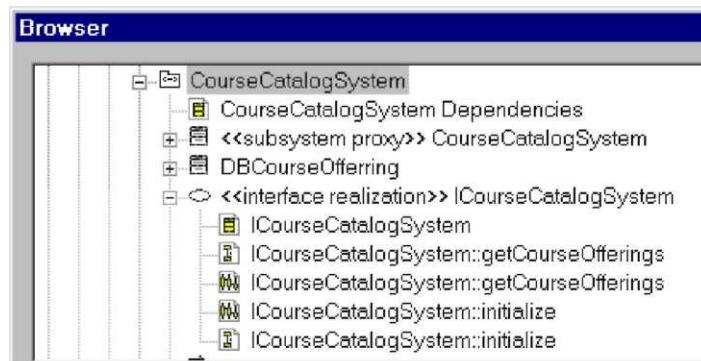


Рис. 3.18. Структура пакета CourseCatalogSystem

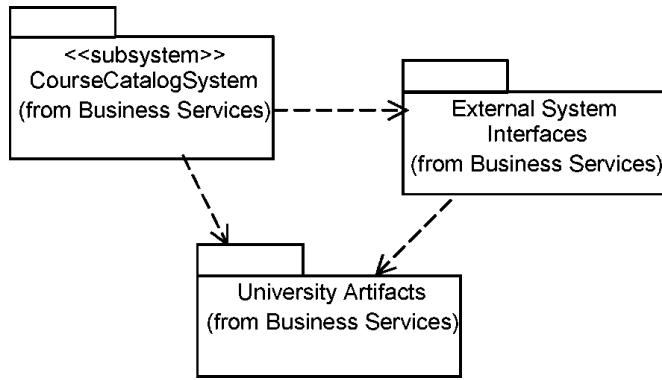


Рис. 3.19. Зависимости между подсистемой и другими пакетами (диаграмма классов CourseCatalogSystem Dependencies)

Чтобы поместить зависимость между пакетами на диаграмму классов:

- 1 . Нажмите кнопку Dependency панели инструментов.
- 2 . Проведите линию зависимости от зависимого пакета к тому, от которого он зависит.

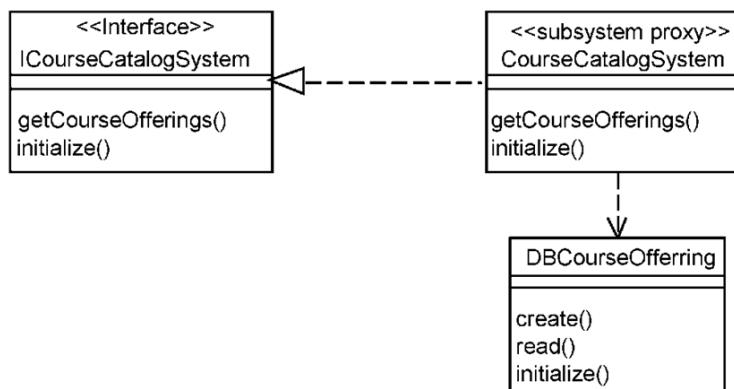


Рис. 3.20. Классы, реализующие интерфейс подсистемы (диаграмма классов ICourseCatalogSystem)

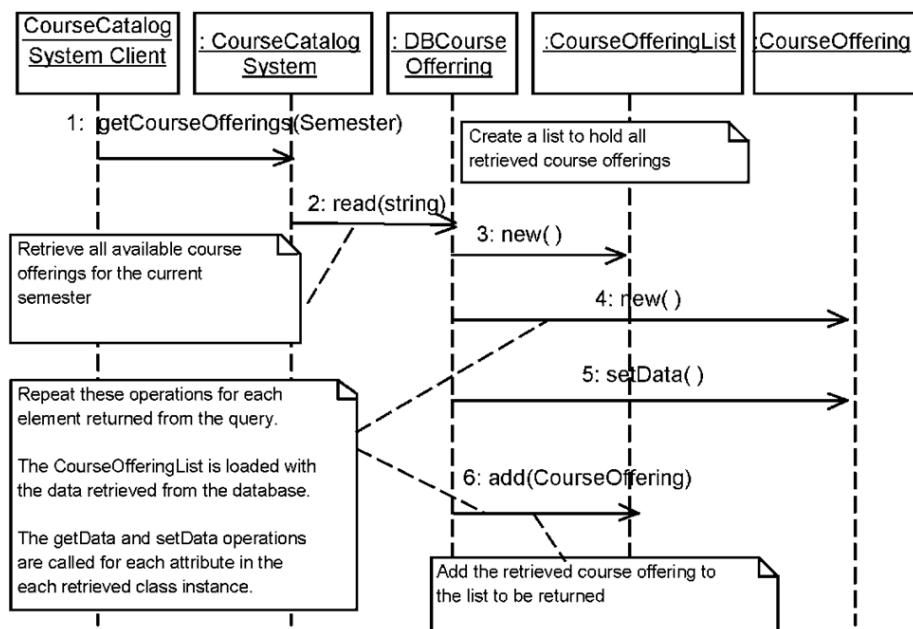


Рис. 3.21. Диаграмма последовательности ICourseCatalogSystem::getCourse Offerings, описывающая взаимодействие элементов при реализации операции интерфейса getCourseOfferings

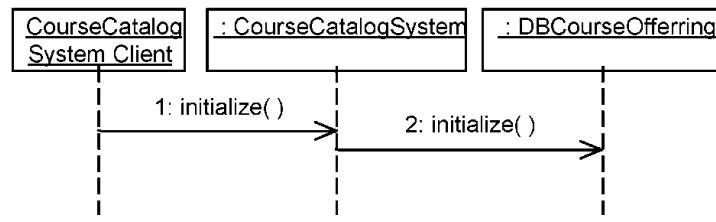


Рис. 3.22. Диаграмма последовательности ICourseCatalogSystem::initialize, описывающая взаимодействие элементов при реализации операции интерфейса initialize

3.6.2. Моделирование распределенной конфигурации системы

Распределенная конфигурация системы моделируется с помощью диаграммы размещения. Ее основные элементы:

- узел (node) - вычислительный ресурс (процессор или другое устройство (дисковая память, контроллеры различных устройств и т. д.). Для узла можно задать выполняющиеся на нем процессы;
- соединение (connection) - канал взаимодействия узлов (сеть).

Пример: сетевая конфигурация системы регистрации (без процессов).

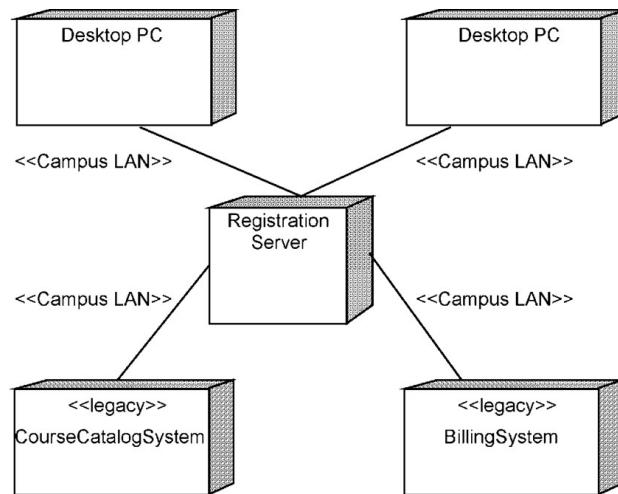


Рис. 3.23. Сетевая конфигурация системы регистрации

Распределение процессов по узлам сети производится с учетом следующих факторов:

- используемые образцы распределения (трехзвенная клиент-серверная конфигурация, «толстый клиент», «тонкий клиент», равноправные узлы (peer-to-peer) и т.д.);
- время отклика;
- минимизация сетевого трафика;
- мощность узла;
- надежность оборудования и коммуникаций.

Пример распределения процессов по узлам приведен на рис. 3.24.

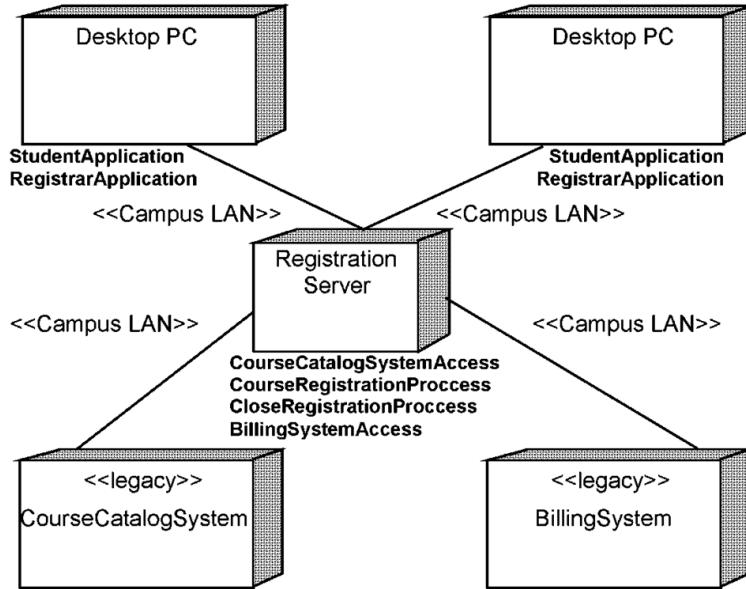


Рис. 3.24. Сетевая конфигурация системы регистрации с распределением процессов по узлам

Упражнение 11. Создание диаграммы размещения системы регистрации

Чтобы открыть диаграмму размещения, надо дважды щелкнуть мышью на представлении Deployment View (представлении размещения) в браузере.

Чтобы поместить на диаграмму процессор:

1. На панели инструментов диаграммы нажмите кнопку Processor.
2. Щелкните на диаграмме размещения в том месте, куда хотите его поместить.
3. Введите имя процессора.

В спецификациях процессора можно ввести информацию о его стереотипе, характеристиках и планировании. Стереотипы применяются для классификации процессоров (например, компьютеров под управлением UNIX или ПК).

Характеристики процессора - это его физическое описание. Оно может, в частности, включать скорость процессора и объем памяти.

Поле планирования (scheduling) процессора содержит описание того, как осуществляется планирование его процессов:

- **Preemptive (с приоритетом).** Высокоприоритетные процессы имеют преимущество перед низкоприоритетными.
- **Non preemptive (без приоритета).** У процессов не имеется приоритета. Текущий процесс выполняется до его завершения, после чего начинается следующий.
- **Cyclic (циклический).** Управление передается между процессами по кругу. Каждому процессу дается определенное время на его выполнение, затем управление переходит к следующему процессу.
- **Executive (исполнительный).** Существует некоторый вычислительный алгоритм, который управляет планированием процессов.
- **Manual (вручную).** Процессы планируются пользователем.

Чтобы назначить процессору стереотип:

- 1 . Откройте окно спецификации процессора.
2. Перейдите на вкладку General.
3. Введите стереотип в поле Stereotype.

Чтобы ввести характеристики и планирование процессора:

- 1 . Откройте окно спецификации процессора.
2. Перейдите на вкладку Detail.
3. Введите характеристики в поле характеристик.
4. Укажите один из типов планирования.

Чтобы показать планирование на диаграмме:

1. Щелкните правой кнопкой мыши на процессоре.

2. В открывшемся меню выберите пункт Show Scheduling.

Чтобы добавить связь на диаграмму:

1 . На панели инструментов нажмите кнопку Connection.

2. Щелкните на узле диаграммы.

3. Проведите линию связи к другому узлу.

Чтобы назначить связи стереотип:

1 . Откройте окно спецификации связи.

2. Перейдите на вкладку General.

3. Введите стереотип в поле Stereotype (Стереотип).

Чтобы добавить процесс:

1 . Щелкните правой кнопкой мыши на процессоре в браузере.

2. В открывшемся меню выберите пункт New > Process.

3. Введите имя нового процесса.

Чтобы показать процессы на диаграмме:

1 . Щелкните правой кнопкой мыши на процессоре.

2. В открывшемся меню выберите пункт Show Processes.

3.6.3. Проектирование классов

Классы анализа преобразуются в проектные классы:

- Проектирование граничных классов - зависит от возможностей среды разработки пользовательского интерфейса (GUI Builder);
- Проектирование классов-сущностей - с учетом соображений производительности (выделение в отдельные классы атрибутов с различной частотой использования);
- Проектирование управляющих классов - удаление классов, реализующих простую передачу информации от граничных классов к сущностям;
- Идентификация устойчивых (persistent) классов, содержащих хранимую информацию.

Обязанности классов, определенные в процессе анализа, преобразуются в операции. Каждой операции присваивается имя, характеризующее ее результат. Определяется полная сигнатура операции: operationName(parameter:class,...):returnType. Создается краткое описание операции, включая смысл всех ее параметров. Определяется видимость операции: public, private, protected. Определяется область действия (scope) операции: экземпляр или классификатор.

Определяются (уточняются) атрибуты классов:

- Кроме имени, задается тип и значение по умолчанию (необязательное): attributeName:type = Default;
- Учитываются соглашения по именованию атрибутов, принятые в проекте и языке реализации;
- Задается видимость атрибутов: public, private, protected;
- При необходимости определяются производные (вычисляемые) атрибуты.

Упражнение 12. Определение атрибутов и операций для класса Student

Чтобы задать тип данных, значение по умолчанию и видимость атрибута:

1 . Щелкните правой кнопкой мыши на атрибуте в браузере.

2. В открывшемся меню выберите пункт Open Specification.

3. Укажите тип данных в раскрывающемся списке типов или введите собственный тип данных.

4. В поле Initial Field (Первоначальное значение) введите значение атрибута по умолчанию.

5. В поле Export Control выберите видимость атрибута: Public, Protected, Private или Implementation. По умолчанию видимость всех атрибутов соответствует Private.

<pre> <<entity>> Student. - name : string - address : string <<class>> - nextAvailID : int - studentID : int - dateofBirth : Date + getTuition() : double + addSchedule(theSchedule : Schedule) + getSchedule(forSemester : Semester) : Schedule + deleteSchedule(forSemester : Semester) + hasPrerequisites(forCourseOffering : CourseOffering) : boolean # passed(theCourseOffering : CourseOffering) : boolean <<class>> + getNextAvailID() : int + getStudentID() : int + getName() : string + getAddress() : string </pre>

Рис. 3.25. Класс Student с полностью определенными операциями и атрибутами

Чтобы изменить нотацию для обозначения видимости:

1. В меню модели выберите пункт Tools > Options.
2. Перейдите на вкладку Notation.
3. Пометьте контрольный переключатель Visibility as Icons, чтобы использовать нотацию Rose, или снимите пометку, чтобы использовать нотацию UML.

Примечание. Изменение значения этого параметра приведет к смене нотации только для новых диаграмм и не затронет уже существующие диаграммы.

Чтобы задать тип возвращаемого значения, стереотип и видимость операции:

- 1 . Щелкните правой кнопкой мыши на операции в браузере.
2. Откройте окно спецификации класса этой операции.
3. Укажите тип возвращаемого значения в раскрывающемся списке или введите свой тип.
4. Укажите стереотип в соответствующем раскрывающемся списке или введите новый.
5. В поле Export Control укажите значение видимости операции: Public, Protected, Private или Implementation. По умолчанию видимость всех операций установлена в public.

Чтобы добавить к операции аргумент:

- 1 . Откройте окно спецификации операции.
2. Перейдите на вкладку Detail.
3. Щелкните правой кнопкой мыши в области аргументов, в открывшемся меню выберите Insert.
4. Введите имя аргумента.
5. Щелкните на колонке Data type и введите туда тип данных аргумента.
6. Если надо, щелкните на колонке default и введите значение аргумента по умолчанию.

Определение состояний для классов моделируется с помощью диаграмм состояний.

Диаграммы состояний создаются для описания объектов с высоким уровнем динамического поведения.

В качестве примера рассмотрим поведение объекта класса CourseOffering. Он может находиться в открытом состоянии (возможно добавление нового студента) или в закрытом состоянии (максимальное количество студентов уже записалось на курс). Таким образом, конкретное состояние зависит от количества студентов, связанных с объектом CourseOffering. Рассматривая каждый вариант использования, можно выделить еще два состояния: инициализация (до начала регистрации студентов на курс) и отмена (курс исключается из расписания).

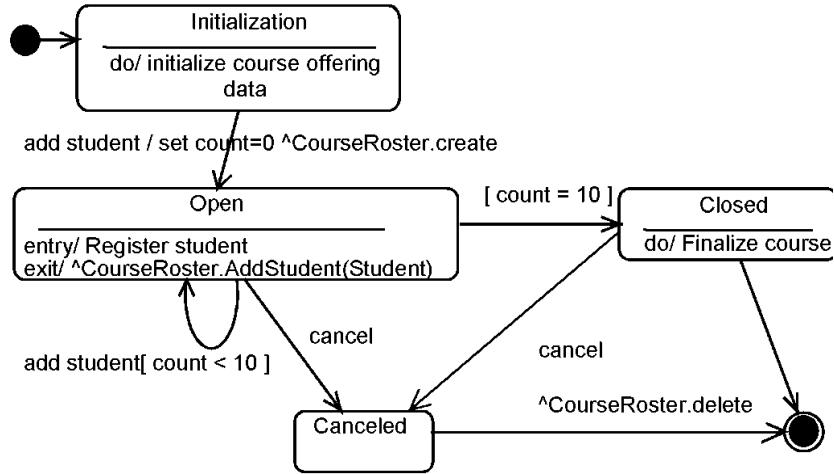


Рис. 3.26. Диаграмма состояний для класса CourseOffering

Упражнение 13. Создание диаграммы состояний для класса CourseOffering

Для создания диаграммы состояний:

- 1 . Щелкните правой кнопкой мыши в браузере на нужном классе.
2. В открывшемся меню выберите пункт New > Statechart Diagram.

Чтобы добавить состояние:

1. На панели инструментов нажмите кнопку State
2. Щелкните мышью на диаграмме состояний в том месте, куда хотите его поместить.

Все элементы состояния можно добавить с помощью вкладки Detail окна спецификации состояния.

Чтобы добавить деятельность:

1. Откройте окно спецификации требуемого состояния.
2. Перейдите на вкладку Detail.
3. Щелкните правой кнопкой мыши на окне Actions.
4. В открывшемся меню выберите пункт Insert.
5. Дважды щелкните на новом действии.
6. Введите действие в поле Actions.
7. В окне When укажите Do, чтобы сделать новое действие деятельностью.

Чтобы добавить входное действие, в окне When укажите On Entry.

Чтобы добавить выходное действие, в окне When укажите On Exit.

Чтобы послать событие:

- 1 . Откройте окно спецификации требуемого состояния.
2. Перейдите на вкладку Detail.
3. Щелкните правой кнопкой мыши на окне Actions.
4. В открывшемся меню выберите пункт Insert.
5. Дважды щелкните на новом действии.
6. В качестве типа действия укажите Send Event.
7. В соответствующие поля введите событие (event), аргументы (arguments) и целевой объект (Target).

Чтобы добавить переход:

- 1 . Нажмите кнопку Transition панели инструментов.
2. Щелкните мышью на состоянии, откуда осуществляется переход.
3. Проведите линию перехода до того состояния, где он завершается.

Чтобы добавить рефлексивный переход:

- 1 . Нажмите кнопку Transition to Self панели инструментов.
2. Щелкните на том состоянии, где осуществляется рефлексивный переход.

Чтобы добавить событие, его аргументы, ограждающее условие и действие:

- 1 . Дважды щелкните на переходе, чтобы открыть окно его спецификации.
2. Перейдите на вкладку General.
3. Введите событие в поле Event.
4. Введите аргументы в поле Arguments.
5. Введите ограждающее условие в поле Condition.
6. Введите действие в поле Action.

Чтобы отправить событие:

1. Дважды щелкните на переходе, чтобы открыть окно его спецификации.
2. Перейдите на вкладку Detail.
3. Введите событие в поле Send Event.
4. Введите аргументы в поле Send Arguments.
5. Задайте цель в поле Send Target.

Для указания начального или конечного состояния:

- 1 . На панели инструментов нажмите кнопку Start State или End State.
2. Щелкните мышью на диаграмме состояний в том месте, куда хотите поместить состояние.

Уточнение ассоциаций: некоторые ассоциации (семантические, структурные, устойчивые связи по данным) могут быть преобразованы в зависимости (неструктурные, временные связи, отражают видимость), а агрегации - в композиции.

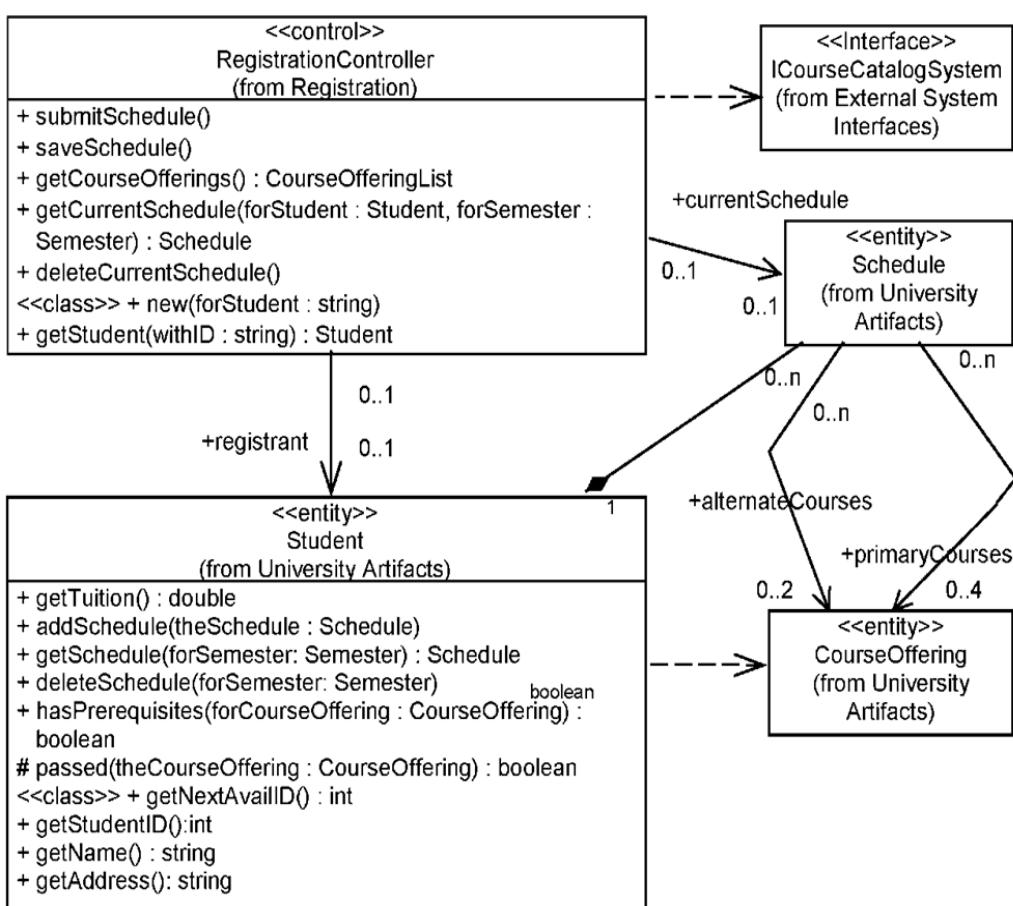


Рис. 3.27. Пример преобразования ассоциаций и агрегаций

Чтобы установить преобразовать агрегацию в композицию:

- 1 . Щелкните правой кнопкой мыши на том конце агрегации, который упирается в класс-часть (на рис.3.27 - Schedule).
2. В открывшемся меню выберите пункт Containment.
3. Укажите метод включения By Value.

Примечание. Значение By Value предполагает, что целое и часть создаются и разрушаются одновременно, что соответствует композиции. Агрегация (By Reference) предполагает, что целое и часть создаются и разрушаются в разное время.

Уточнение обобщений: в случае ситуации с миграцией подклассов (студент может переходить с очной формы обучения на вечернюю) иерархия наследования реализуется так, как показано на рис. 3.28 . Такое решение повышает устойчивость системы (не нужно модифицировать описание объекта).

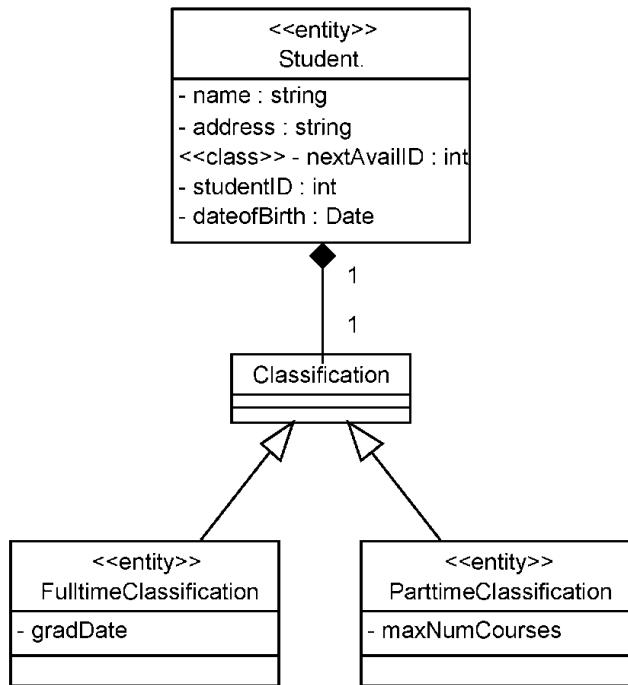


Рис. 3.28 Преобразование обобщения

3.6.4. Проектирование баз данных

Проектирование реляционных баз данных выполняется с использованием средства Data Modeler. Его работа основана на известном механизме отображения объектной модели в реляционную. Результатом является построение диаграммы «сущность-связь» и последующая генерация описания БД на SQL.

Упражнение 14. Проектирование реляционной базы данных

Проектирование БД состоит из следующих шагов:

Создание нового компонента - базы данных:

- 1 . Щелкните правой кнопкой мыши на представлении компонентов.
- 2 . В открывшемся меню выберите пункт Data Modeler > New > Database.
- 3 . Откройте окно спецификации вновь созданного компонента DB_0 и в списке Target выберите Oracle 8.x.

Определение устойчивых (persistent) классов:

- 1 . Откройте окно спецификации класса Student в пакете University Artifacts.
- 2 . Перейдите на вкладку Detail.
- 3 . Установите значение переключателя Persistence в Persistent.
- 4 . Проделайте такие же действия для классов Classification, FulltimeClassification и ParttimeClassification.
- 5 . Откройте класс Student в браузере, нажав « + ».
- 6 . Щелкните правой кнопкой мыши на атрибуте studentID.
- 7 . В открывшемся меню выберите пункт Data Modeler > Part of Object Identity (указание атрибута в качестве части первичного ключа).

Примечание. Шаги 5, 6 и 7 можно выполнять в Rational Rose, начиная с версии 2001 .

Создание схемы БД:

- 1 . Щелкните правой кнопкой мыши на пакете University Artifacts.
 2. В открывшемся меню выберите пункт Data Modeler > Transform to Data Model.
 3. В появившемся окне в списке Target Database укажите DB_0 и нажмите OK. В результате в логическом представлении появится новый пакет Schemas.
 4. Откройте пакет Schemas и щелкните правой кнопкой мыши на пакете <<Schema>> S_0.
 5. В открывшемся меню выберите пункт Data Modeler > New > Data Model Diagram.
 6. Откройте пакет, затем откройте вновь созданную диаграмму «сущность-связь» NewDiagram и перенесите на нее все классы-таблицы, находящиеся в пакете <<Schema>> S_0.
- Получившаяся диаграмма показана на рис. 3.29.

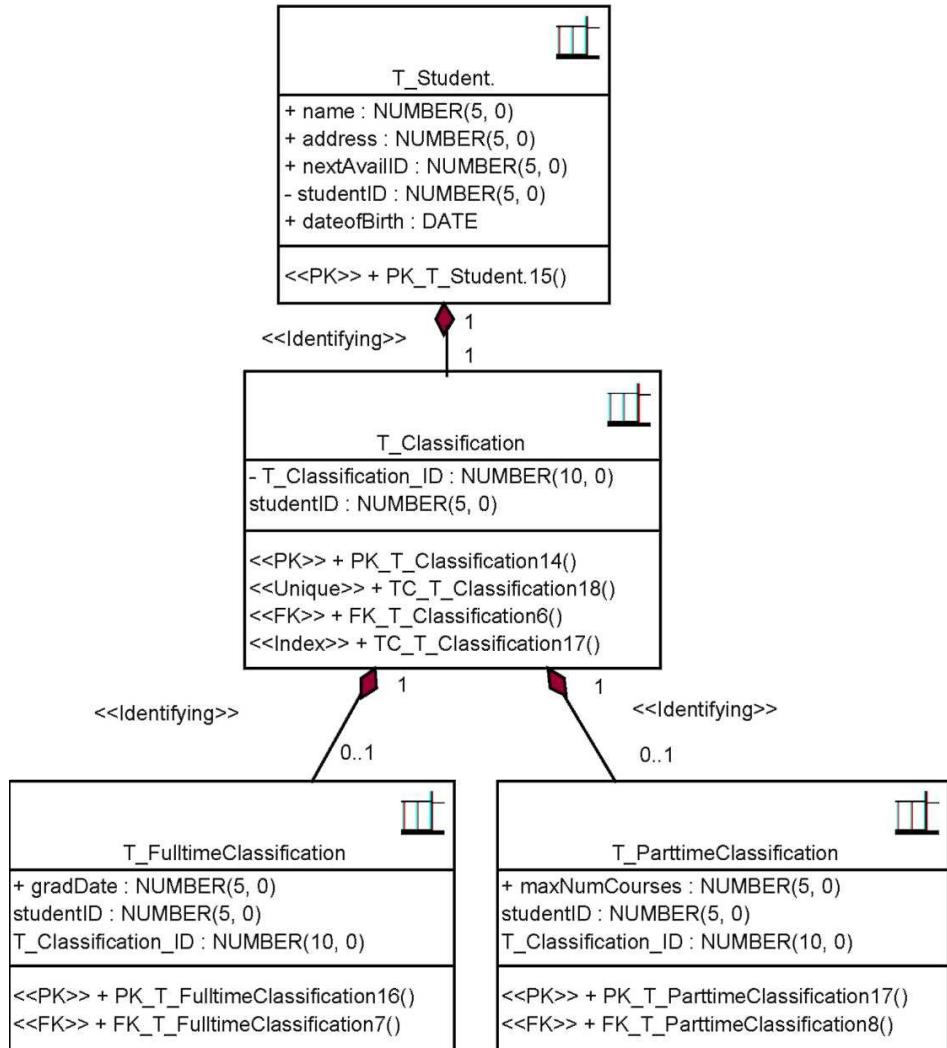


Рис. 3.29. Диаграмма «сущность-связь»

Генерация описания БД на SQL:

- 1 . Щелкните правой кнопкой мыши на пакете <<Schema>> S_0.
2. В открывшемся меню выберите пункт Data Modeler > Forward Engineer.
3. В открывшемся окне мастера Forward Engineering Wizard нажмите Next.
4. Отметьте все флажки генерации DDL и нажмите Next.
5. Укажите имя и расположение текстового файла с результатами генерации и нажмите Next.
6. После завершения генерации откройте созданный текстовый файл и просмотрите результаты.

3.7. Реализация системы

3.7.1. Создание компонентов

В Rational Rose диаграммы компонентов создаются в представлении компонентов системы. Отдельные компоненты можно создавать непосредственно на диаграмме, или перетаскивать их туда из браузера.

Упражнение 15. Создание компонентов

Выберем в качестве языка программирования C++ и для класса Student создадим соответствующие этому языку компоненты.

Создание диаграммы компонентов:

- 1 . Дважды щелкните мышью на главной диаграмме компонентов в представлении компонентов.
2. На панели инструментов нажмите кнопку Package Specification.
3. Поместите спецификацию пакета на диаграмму.
4. Введите имя спецификации пакета Student и укажите в окне спецификации язык C++.
5. На панели инструментов нажмите кнопку Package Body.
6. Поместите тело пакета на диаграмму.
7. Введите имя тела пакета Student и укажите в окне спецификации язык C++.
8. На панели инструментов нажмите кнопку Dependency.
9. Проведите линию зависимости от тела пакета к спецификации пакета.

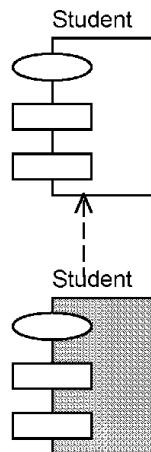


Рис. 3.30. Диаграмма компонентов

Соотнесение классов с компонентами:

1. В логическом представлении браузера найдите класс Student.
2. Перетащите этот класс на спецификацию пакета компонента Student в представлении компонентов браузера. В результате класс Student будет соотнесен со спецификацией пакета компонента Student.
3. Перетащите класс Student на тело пакета компонента Student в представлении компонентов браузера. В результате класс Student будет соотнесен с телом пакета компонента Student.

3.7.2. Генерация кода

Процесс генерации кода состоит из четырех основных шагов:

- 1 . Проверка корректности модели.
2. Установка свойств генерации кода.
3. Выбор класса, компонента или пакета.
4. Генерация кода.

Для проверки модели:

- 1 . Выберите в меню Tools > Check Model.

2. Проанализируйте все найденные ошибки в окне журнала.

К наиболее распространенным ошибкам относятся такие, например, как сообщения на диаграмме последовательности или кооперативной диаграмме, не соотнесённые с операцией, либо объекты этих диаграмм, не соотнесённые с классом.

С помощью пункта меню Check Model можно выявить большую часть неточностей и ошибок в модели. Пункт меню Access Violations позволяет обнаруживать нарушения правил доступа, возникающие тогда, когда существует связь между двумя классами разных пакетов, но связи между самими пакетами нет.

Чтобы обнаружить нарушение правил доступа:

1. Выберите в меню Report > Show Access Violations.
2. Проанализируйте все нарушения правил доступа в окне.

Можно установить несколько параметров генерации кода для классов, атрибутов, компонентов и других элементов модели. Этими свойствами определяется способ генерации программ. Для каждого языка в Rose предусмотрен ряд определенных свойств генерации кода. Перед генерацией кода рекомендуется анализировать эти свойства и вносить необходимые изменения.

Для анализа свойств генерации кода выберите Tools > Options, а затем вкладку соответствующего языка. В окне списка можно выбрать класс, атрибут, операцию и другие элементы модели. Для каждого языка в этом списке указаны свои собственные элементы модели. При выборе разных значений на экране появляются разные наборы свойств.

Любые изменения, вносимые в набор свойств в окне Tools > Options, воздействуют на все элементы модели, для которых используется данный набор.

Иногда нужно изменить свойства генерации кода для одного класса, атрибута, одной операции и т.д. Для этого отройте окно спецификации элемента модели. Выберите вкладку языка (C++, Java, ...) и измените свойства здесь. Все изменения, вносимые в окне спецификации элемента модели, оказывают влияние только на этот элемент.

При генерации кода за один раз можно создать класс, компонент или целый пакет. Код генерируется с помощью диаграммы или браузера. При генерации кода из пакета можно выбрать или пакет логического представления на диаграмме классов, или пакет представления компонентов на диаграмме компонентов. При выборе пакета логического представления генерируются все классы этого пакета. При выборе пакета представления компонентов генерируются все компоненты этого пакета.

После выбора класса или компонента на диаграмме выберите в меню соответствующий вариант генерации кода. Сообщения об ошибках, возникающих в процессе генерации кода, будут появляться в окне журнала.

Во время генерации кода Rose выбирает информацию из логического и компонентного представлений модели и генерирует большой объем «скелетного» (skeletal) кода:

- **Классы.** Генерируются все классы модели.
- **Атрибуты.** Код включает атрибуты каждого класса, в том числе видимость, тип данных и значение по умолчанию.
- **Сигнатуры операций.** Код содержит определения операций со всеми параметрами, типами данных параметров и типом возвращаемого значения операции.
- **Связи.** Некоторые из связей модели вызывают создание атрибутов при генерации кода.
- **Компоненты.** Каждый компонент реализуется в виде соответствующего файла с исходным кодом.

Упражнение 16. Генерация кода C++

- 1 . Откройте диаграмму компонентов системы.
2. Выберите все объекты на диаграмме компонентов.
3. Выберите Tools > C++ > Code Generation в меню.
4. Выполните генерацию кода.
5. Просмотрите результаты генерации (меню Tools > C++ > Browse Header и Tools > C++ > Browse Body).

Глава 4. Варианты заданий для самостоятельной работы

В каждом из предложенных вариантов требуется при помощи CASE-средства Rational Rose построить модель программного обеспечения. Процесс создания модели должен проходить так, как это описано в главе 3. Должны быть выполнены следующие действия:

- 1) составление глоссария проекта;
- 2) создание модели вариантов использования;
- 3) анализ вариантов использования;
- 4) проектирование системы;
- 5) реализация системы.

После выполнения третьего этапа модель должна удовлетворять перечисленным ниже требованиям. Глоссарий проекта должен иметь вид таблицы и храниться в отдельном файле. На диаграммах вариантов использования каждое действующее лицо (actor) и вариант использования должны сопровождаться описанием. Эти описания должны быть составлены на русском языке. Описание действующего лица должно коротко (в одну-две строки) сообщать о роли данного лица. Описание варианта использования должно включать в себя пояснение, предусловие, потоки событий (основной и альтернативные, если таковые есть) и постусловие. Описания представляют собой либо присоединенные текстовые файлы, либо текст, введенный в поле Documentation спецификации соответствующего элемента диаграммы. Диаграммы взаимодействия, соответствующие потокам событий вариантов использования, должны содержать необходимые пояснения.

При проектировании системы требуется:

- создать иерархию классов системы;
- разместить классы по пакетам (использовать деление: пользовательский интерфейс - управление - данные; или другое в зависимости от постановки задачи);
- связать объекты с классами, сообщения на диаграммах взаимодействия - с операциями;
- каждый класс снабдить описанием, которое должно включать в себя краткое описание (ответственность класса), описание атрибутов в виде таблицы (имя, описание, тип), таблицу с описанием операций (имя, описание, сигнатура);
- для классов указать стереотипы;
- построить диаграммы классов системы, отображающие связи между классами;
- для описания поведения экземпляров отдельных классов построить диаграммы состояний;
- разработать (если это требуется вариантом задания) схему базы данных и отобразить ее на диаграмме «сущность - связь».

При реализации системы необходимо построить диаграммы компонентов для каждого пакета и для системы в целом. Также следует разработать диаграмму размещения. В зависимости от варианта задания диаграмма размещения должна показывать расположение компонентов в распределенном приложении или связи между встроенным процессором и устройствами. Должна быть произведена проверка корректности модели и автоматическая генерация кода средствами Rational Rose.

Ниже перечислены варианты заданий.

4.1. Цифровой диктофон

Требуется разработать средствами Rational Rose модель программного обеспечения, управляющего работой цифрового диктофона.

Цифровой диктофон - это бытовое электронное устройство, предназначенное для записи и воспроизведения речи. Звуковые сообщения записываются через встроенный микрофон и сохраняются в памяти устройства. Сообщения воспроизводятся через встроенный громкоговоритель. Работа устройства осуществляется под управлением центрального процессора. Примерный внешний вид устройства изображен на рисунке 4.1 .

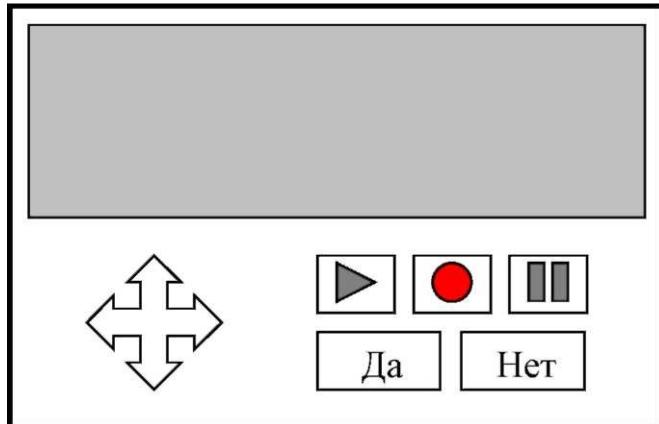


Рис. 4.1. Внешний вид диктофона

Диктофон хранит до 10 звуковых сообщений. Длина каждого сообщения ограничена размером свободной памяти. Диктофон осуществляет прямой (по номеру сообщения) доступ к любому сообщению из памяти. Пользователь имеет возможность воспроизводить сообщения, хранящиеся в памяти диктофона, стирать их, записывать новые. Исполнителем должна быть разработана схема базы данных для хранения сообщений в памяти диктофона.

Интерфейс с пользователем осуществляется при помощи экранного меню и управляющих кнопок на корпусе диктофона. При помощи кнопок-стрелок осуществляется навигация по пунктам меню. Кнопки «Да», «Нет» служат для подтверждения или отмены пользователем выбора той или иной опции меню (структуру меню исполнитель должен разработать самостоятельно). Имеются также кнопки «Воспроизведение», «Пауза» и «Запись» для работы со звуковыми сообщениями.

Во время записи сообщения на экране отображается время, в течение которого ведется запись, при воспроизведении - длительность воспроизведенной части сообщения.

Если диктофон не используется, через 30 секунд он автоматически переходит в режим сбережения энергии. В этом режиме никакие операции над звуковыми сообщениями не возможны. Энергия расходуется только на сохранение памяти диктофона в неизменном состоянии. Переход из режима сбережения энергии в обычный режим осуществляется при нажатии пользователем любой кнопки.

В диктофоне имеется датчик уровня заряда батарей. При падении уровня заряда ниже установленного предела диктофон автоматически переходит в режим сбережения энергии (независимо от того используется он в данный момент или нет). Переход в обычный режим становится возможным только после восстановления нормального уровня заряда батарей.

4.2. Торговый автомат

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного процессора универсального торгового автомата.

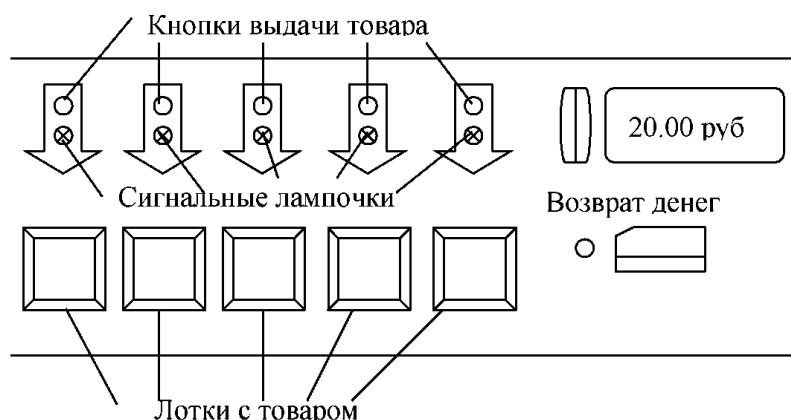


Рис. 4.2. Лицевая панель торгового автомата

Внешний вид автомата изображен на рисунке 4.2. В автомате имеется пять лотков для хранения и выдачи товаров. Загрузка товаров на лотки осуществляется обслуживающим персоналом. Автомат следит за наличием товара. Если какой-либо товар распродан, автомат отправляет сообщение об этом на станцию обслуживания и информирует покупателей (зажигается красная лампочка рядом с лотком данного товара).

Автомат принимает к оплате бумажные купюры и монеты. Специальный индикатор высвечивает текущую сумму денег, принятых автоматом к оплате. После ввода денег клиент нажимает на кнопку выдачи товара. Выдача товара производится только в том случае, если введенная сумма денег соответствует цене товара. Товар выдается поштучно. При нажатии на кнопку «Возврат» клиенту возвращаются все принятые от него к оплате деньги. Возврат денег не производился после выдачи товара. Автомат должен корректно работать при одновременном нажатии на кнопки выдачи товара и возврата денег.

В специальном отделении автомата, закрываемом замком, есть «секретная кнопка», которая используется обслуживающим персоналом для выемки выручки. При нажатии на эту кнопку открывается доступ к ящику с деньгами.

Автомат получает со станции обслуживания данные о товарах и хранит их в своей памяти. Данные включают в себя цену, наименование товара, номер лотка, на котором находится товар и количество товара на лотке. Вариант задания включает в себя разработку схемы базы данных о товарах.

4.3. Табло на станции метро

Требуется разработать средствами Rational Rose модель программного обеспечения табло для информационной службы метрополитена.

Табло расположены на каждой станции метро. Они работают под управлением единого пункта управления (ПУ) информационной службы метро. Табло отображает текущее время (часы, минуты, секунды) и время, прошедшее с момента отправления последнего поезда (минуты, секунды). Момент прибытия и отправления поезда определяется при помощи датчиков, устанавливаемых на путях. Все табло метро синхронизованы, текущее время отсчитывается и устанавливается из центральной службы времени, находящейся на ПУ.

На табло высвечивается конечная станция назначения прибывающего поезда. Эти данные содержатся в расписании движения поездов, которое хранится в памяти табло и периодически обновляется с ПУ.

В «бегущей строке» табло отображается рекламная информация. Память табло хранит до 10 рекламных сообщений. Сообщения отображаются друг за другом с небольшими паузами, циклически. Содержание рекламных сообщений поступает с ПУ.

Дополнительная функция табло - по запросу с ПУ оно пересыпает данные о нарушениях расписания (преждевременных отправлениях поездов или опозданиях).

В ходе выполнения задания должна быть создана схема базы данных для хранения рекламных сообщений, расписания и сведений о нарушении расписаний.

Пояснение: в задании требуется разработать модель ПО только для табло, но не для пункта управления информационной службы.

4.4. Система автоматизации для пункта проката видеокассет

Требуется разработать средствами Rational Rose модель программной системы автоматизации работы пункта проката видеокассет (далее в тексте - системы).

Пункт проката содержит каталог кассет, имеющихся в наличии в данный момент времени. Система поддерживает работу каталога, позволяя служащим проката добавлять новые наименования кассет, удалять старые и редактировать данные о кассетах.

Клиент, обратившийся в пункт, выбирает кассету по каталогу, вносит залог и забирает ее на определенный срок. Срок проката, измеряемый в сутках, оговаривается при выдаче кассеты. Стоимость проката вычисляется системой исходя из тарифа за сутки и срока проката. Клиент возвращает кассету и оплачивает прокат. Если кассета не повреждена, клиенту возвращается залог. Служащий пункта проката регистрирует сдачу кассеты клиенту и ее возврат в системе. Если клиент повредил кассету, то кассета удаляется из каталога, а залог остается в кассе проката.

При необходимости служащий может запросить у системы следующие данные:

- имеется ли в наличии кассета с данным названием;
- когда будет возвращена какая-либо кассета из тех, что сданы в прокат;
- является ли данный клиент постоянным клиентом пункта проката (пользовался ли прокатом 5 или более раз).

Постоянным клиентам предоставляются скидки, а также от них принимаются заявки на пополнение ассортимента кассет. Заявки регистрируются в системе. По ним готовится итоговый отчет, руководствуясь которым, служащие пункта проката обновляют ассортимент кассет.

Необходимо разработать схему базы данных для хранения каталога, учетных записей о прокате кассет и заявок на пополнение ассортимента.

4.5. Мини-АТС

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного микропроцессора учрежденческой мини-АТС (автоматической телефонной станции).

Мини-АТС осуществляет связь между служащими учреждения. Каждый абонент подключен к ней линией связи. Мини-АТС соединяет линии абонентов (осуществляет коммутацию линий). Абоненты имеют номера, состоящие из трех цифр. Специальный номер 9 зарезервирован для внешней связи.

Телефонное соединение абонентов производится следующим образом. Абонент поднимает трубку телефона, и мини-АТС получает сигнал «Трубка». В ответ мини-АТС посыпает сигнал «Тон». Приняв этот сигнал, абонент набирает телефонный номер (посыпает три сигнала «Цифра»). Мини-АТС проверяет готовность вызываемого абонента. Если абонент не готов (его линия занята), мини-АТС посыпает вызывающему абоненту сигнал «Занято». Если абонент готов, мини-АТС посыпает обоим абонентам сигнал «Вызов». При этом телефон вызываемого абонента начинает звонить, а вызывающий абонент слышит в трубке длинные гудки. Вызываемый абонент снимает трубку, и мини-АТС получает от него сигнал «Трубка», после чего осуществляет коммутацию линии. Абоненты обмениваются сигналами «Данные», которые мини-АТС должна передавать от одного абонента к другому. Когда один из абонентов опускает трубку, мини-АТС получает сигнал «Конец» и посыпает другому абоненту сигнал «Тон».

В любой момент абонент может положить трубку, при этом мини-АТС получает сигнал «Конец». После получения этого сигнала сеанс обслуживания абонента завершается.

Если абонент желает соединиться с абонентом за пределами учреждения, то он набирает номер «9». Мини-АТС посыпает по линии, соединяющей с внешней (городской) АТС, сигнал «Трубка» и в дальнейшем служит посредником между телефоном абонента и внешней АТС. Она принимает и передает сигналы и данные между ними, не внося никаких изменений. Единственное исключение касается завершения сеанса. Получив от городской АТС сигнал «Конец», мини-АТС посыпает абоненту сигнал «Тон», и ждет сигнала «Конец» для завершения обслуживания абонента. Если же вызывавший абонент первым вешает трубку, то мини-АТС получает сигнал «Конец», передает его городской АТС и завершает сеанс.

Мини-АТС может получить сигнал «Вызов» от городской АТС. Это происходит, когда нет соединений с внешними абонентами. Сигнал «Вызов» от городской АТС передается абоненту с кодом «000». Только этот абонент может отвечать на внешние звонки.

4.6. Телефон

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного микропроцессора для аппарата учрежденческой телефонной сети.

Аппарат подключен к линии связи, ведущей к мини-АТС. В его задачу входит прием и передача сигналов (в том числе и голосовых данных) мини-АТС. Аппарат имеет кнопочную панель управления, экран для отображения набираемых номеров, звонок и трубку, в которую встроены микрофон и громкоговоритель.

В начальном состоянии трубка телефона повешена, телефон не реагирует на нажатия кнопок. Телефон реагирует только на сигнал «Вызов» от мини-АТС, при этом включается звонок.

При снятии трубки на АТС подается сигнал «Трубка». При получении ответного сигнала «Тон» от АТС телефон воспроизводит звуковой тон «Готов» (длинный непрекращающийся гудок) в трубку. При получении сигнала «Занято», в трубке воспроизводится тон «Занято» (частые короткие гудки).

Пользователь, слыша в трубке тон «Готов», набирает трехзначный номер. Номер может быть набран при помощи кнопок с цифрами или нажатием на специальную кнопку «#». При нажатии на кнопку с цифрой соответствующий ей сигнал «Цифра» передается АТС. Нажатия на кнопки с цифрами после третьего игнорируются. Во время набора номера введенные цифры отображаются на экране. Последний полностью набранный номер запоминается в памяти аппарата для того, чтобы можно было его воспроизвести при нажатии на кнопку «#». При нажатии на эту кнопку номер из памяти аппарата высвечивается на экране, и АТС передается последовательность из трех сигналов «Цифра». В ответ на набранный номер от АТС приходит либо сигнал «Занято», либо сигнал «Вызов». При получении сигнала «Вызов» телефон воспроизводит в трубку длинные гудки до того момента, когда АТС осуществит коммутацию и передаст сигнал «Данные».

Телефон воспроизводит данные, передаваемые с сигналом, в трубку. Ответ пользователя воспринимается микрофоном трубки, преобразуется в сигнал «Данные» и передается АТС. Обмен данными прерывается, если повешена трубка одного из телефонов, участвующих в обмене. О том, что трубку повесил вызываемый абонент, сообщает сигнал «Занято», посыпаемый АТС. После того, как трубка аппарата была повешена, телефон посыпает АТС сигнал «Конец», и телефон переходит в начальное состояние.

4.7. Стиральная машина

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного микропроцессора стиральной машины.

Машина предназначена для автоматической стирки белья. Машина включает в себя следующие устройства: бак для белья, клапаны для забора и слива воды, мотор, устройство подогрева воды, термометр, таймер, дверца для доступа в бак, несколько емкостей для различных моющих средств, панель управления с кнопками и индикатором. В памяти машины хранятся 5 программ стирки, заданные изготовителем. Пользователи не могут вносить в них изменения. Каждая программа определяет температуру воды, длительность стирки, используемые моющие средства (номер емкости и время подачи), скорость вращения бака во время стирки и отжима. Вариант задания предусматривает разработку схемы базы данных для хранения программ стирки в памяти машины.

Для использования машины необходимо открыть дверцу, поместить белье в бак, поместить моющие средства в емкости, закрыть дверцу, выбрать программу стирки и нажать на кнопку «Пуск». Перед тем как приступить к стирке машина открывает клапан для забора воды, набирает необходимое количество воды, после чего закрывает клапан. Далее, машина действует по выбранной пользователем программе:

- 1) Подогревает, если необходимо, воду до нужной температуры.
- 2) Включает таймер и запускает вращение бака для стирки.
- 3) По таймеру подает в бак моющие средства, предусмотренные программой.
- 4) По окончании стирки сливает воду и запускает отжим.

Во время работы машины на индикаторе высвечивается время, прошедшее с момента запуска (минуты и секунды), текущий режим работы (стирка или отжим), номер текущей программы стирки. В целях безопасности дверца бака блокируется до окончания стирки. Машина не воспринимает нажатий на кнопки, за исключением одной - пользователь имеет возможность в любой момент нажать на кнопку «Останов», чтобы принудительно остановить стирку и слить воду.

4.8. Таксофон

Требуется разработать средствами Rational Rose модель встроенной системы управления работой таксофона городской телефонной сети.

Таксофон предназначен для оказания платных услуг телефонной связи. Он подключен к линии связи. В нем имеется кнопочная панель, дисплей, трубка со встроенным микрофоном и громкоговорителем, приемник карт - устройство для считывания телефонных карт, используемых для оплаты разговора.

В начальном состоянии трубка таксофона повешена, дисплей потушен, таксофон не реагирует на нажатия кнопок и какие-либо сигналы из линии. При снятии трубки таксофон выдает на дисплей сообщение «Вставьте карту» и ожидает, когда пользователь вставит карту в приемник. Дальнейшее функционирование таксофона осуществляется только при вставленной карте. Если карту вынимают, таксофон возвращается к началу и выдает сообщение о необходимости вставить карту. При попадании карты в приемник производится считывание информации с карты. Если кредит исчерпан или карта не пригодна (не удается узнать кредит), то таксофон выдает соответствующее сообщение на дисплей таксофона. Если карта может быть использована для оплаты, то на дисплей выдается количество «единиц» на карте, и на телефонную станцию (АТС) подается сигнал «Трубка». При получении ответного сигнала «Тон» из линии таксофон воспроизводит звуковой тон «Готов» (длинный непрекращающийся гудок) в трубку. При получении сигнала «Занято», в трубке воспроизводится тон «Занято» (короткие гудки).

После получения от АТС сигнала «Тон» от пользователя принимаются семизначный номер вызываемого абонента, остальные нажатия на кнопки игнорируются. Когда пользователь нажимает на кнопку с цифрой соответствующий ей сигнал «Цифра» передается АТС. Во время набора номера введенные цифры отображаются на дисплее. В ответ на набранный номер от АТС приходит либо сигнал «Занято», либо сигнал «Вызов». При получении сигнала «Вызов» таксофон воспроизводит в трубку длинные гудки до того момента, когда АТС осуществит коммутацию и передаст сигнал «Данные». Таксофон воспроизводит данные, передаваемые с сигналом, в трубку. При получении данных из трубки, аппарат преобразует их в сигнал «Данные» и передает их АТС. Во время разговора на дисплее ведется отсчет времени и уменьшается кредит на телефонной карте - каждые 15 секунд вычитается четверть «единицы». Обмен данными прерывается, в следующих случаях:

- исчерпан кредит;
- карта вынута из приемника;
- от АТС пришел сигнал «Занято»;
- повешена трубка таксофона.

Если трубка была повешена, аппарат посыпает в линию сигнал «Конец» и выдает на дисплей сообщение «Выньте карту». После извлечения карты из приемника таксофон переходит в начальное состояние.

4.9. Банкомат

Требуется разработать средствами Rational Rose модель программного обеспечения банкомата. Банкомат - это автомат для выдачи наличных денег по кредитным пластиковым карточкам. В его состав входят следующие устройства: дисплей, панель управления с кнопками, приемник кредитных карт, хранилище денег и лоток для их выдачи, хранилище конфискованных кредитных карт, принтер для печати справок.

Банкомат подключен к линии связи для обмена данных с банковским компьютером, хранящим сведения о счетах клиентов.

Обслуживание клиента начинается с момента помещения пластиковой карточки в банкомат. После распознавания типа пластиковой карточки, банкомат выдает на дисплей приглашение ввести персональный код. Персональный код представляет собой четырехзначное число. Затем банкомат проверяет правильность введенного кода. Если код указан неверно, пользователю предоставляются еще две попытки для ввода правильного кода. В случае повторных неудач карта перемещается в хранилище карт, и сеанс обслуживания заканчивается. После ввода правильного кода банкомат предлагает пользователю выбрать операцию. Клиент может либо снять наличные со счета, либо узнать остаток на его счету.

При снятии наличных со счета банкомат предлагает указать сумму (10, 50, 100, 200, 500, 1000 рублей). После выбора клиентом суммы банкомат запрашивает, нужно ли печатать справку по операции. Затем банкомат посылает запрос на снятие выбранной суммы центральному компьютеру банка. В случае получения разрешения на операцию, банкомат проверяет, имеется ли требуемая сумма в его хранилище денег. Если он может выдать деньги, то на дисплей выводится сообщение «Выньте карту». После удаления карточки из приемника, банкомат выдает указанную сумму в лоток выдачи. Банкомат печатает справку по произведенной операции, если она была затребована клиентом.

Если клиент хочет узнать остаток на счету, то банкомат посыпает запрос центральному компьютеру банка и выводит сумму на дисплей. По требованию клиента печатается и выдается соответствующая справка.

В специальном отделении банкомата, закрываемом замком, есть «секретная кнопка», которая используется обслуживающим персоналом для загрузки денег. При нажатии на эту кнопку открывается доступ к хранилищу денег и конфискованным кредитным картам.

4.10. Холодильник

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного процессора холодильника. Холодильник состоит из нескольких холодильных камер для хранения продуктов. В каждой холодильной камере имеется регулятор температуры, мотор, термометр, индикатор, таймер, датчик открытия двери камеры и устройство для подачи звуковых сигналов.

При помощи терморегулятора устанавливается максимально допустимая температура в данной камере. Мотор предназначен для поддержания низкой температуры. Термометр постоянно измеряет температуру внутри камеры, а индикатор температуры, расположенный на дверце, постоянно высвечивает ее значение. При повышении температуры выше предела, определяемого текущим положением регулятора, включается мотор. При снижении температуры ниже некоторого другого значения, связанного с первым, мотор отключается.

Доступ в камеру осуществляется через дверцу. Если дверь холодильной камеры открыта в течение слишком долгого времени, подается звуковой сигнал. Звуковой сигнал также подается в любых нештатных ситуациях (например, при поломке мотора).

Холодильник ведет электронный журнал, в котором отмечаются все происходящие события:

- изменение положения терморегулятора камеры;
- включение и отключение мотора;
- доступ в камеру;
- нештатные ситуации.

Вариант задания предусмотрена разработка схемы базы данных для хранения журнала событий холодильника. Содержимое журнала может быть передано в компьютер, подсоединенный к специальному гнезду на корпусе холодильника.

4.11. Кодовый замок

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного микропроцессора для кодового замка, регулирующего доступ в помещение.

Кодовый замок состоит из панели с кнопками (цифры «0»...«9», кнопка «Вызов», кнопка «Контроль»), цифрового дисплея, электромеханического замка, звонка. Панель с кнопками устанавливается с наружной стороны двери, замок устанавливается с внутренней стороны двери, звонок устанавливается внутри охраняемого помещения.

В обычном состоянии замок закрыт. Доступ в помещение осуществляется после набора кода доступа, состоящего из четырех цифр. Во время набора кода введенные цифры отображаются на дисплее. Если код набран правильно, то замок открывается на некоторое время, после чего дверь снова закрывается. Содержимое дисплея очищается.

Кнопка «Вызов» используется для подачи звукового сигнала внутри помещения. Кнопка «Контроль» используется для смены кодов. Смена кода доступа осуществляется следующим образом. При открытой двери нужно набрать код контроля, состоящий из четырех цифр, и новый код доступа. Для смены кода контроля нужно при открытой двери и нажатой кнопке «Вызов» набрать код контроля, после чего - новый код контроля.

4.12. Турникет метро

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного процессора турникета для метрополитена.

При помощи турникета контролируется проход пассажиров в метро и взимается входная плата. Турникет имеет приемник карт, устройство для перекрывания доступа, таймер, три оптических

датчика для определения прохода пассажира, устройство подачи звуковых сигналов, индикаторы «Проход» и «Стоп».

В начальном состоянии турникета зажжен индикатор «Стоп», индикатор «Проход» потушен. Если один из датчиков посыпает сигнал, то проход через турникет сразу же перекрывается, и подается предупредительный звуковой сигнал. Для прохода пассажир должен поместить карту в приемник карт. Турникет считывает с нее данные: срок годности карты и количество «единиц» на ней. Если данные не удается считать, или карта просрочена, или заблокирована, то карта возвращается пассажиру, и турникет остается в исходном состоянии. В другом случае с карты списывается одна «единица», карта возвращается из приемника, индикатор «Стоп» гаснет, зажигается индикатор «Проход», и пассажир может пройти через турникет. Получив от одного из датчиков сигнал, турникет ожидает время, отведенное на проход пассажира (5 секунд), после чего он возвращается в начальное состояние.

Наличие трех датчиков в турникеете гарантирует, что при проходе пассажира хотя бы один из них подаст сигнал (датчики невозможно перешагнуть, перепрыгнуть и т.д.). Во время прохода пассажира возможна ситуация, когда все три датчика посыпают сигналы. В этом случае принимается только первый сигнал и от момента его приема отсчитывается положенное время. Остальные сигналы игнорируются.

Турникет заносит в свою память время всех оплаченных проходов. В конце рабочего дня он передает всю информацию, накопленную за день, в АСУ метрополитена.

В ходе выполнения этого варианта задания должна быть разработана схема базы данных о проходах через турникет.

4.13. Система учета товаров

Требуется разработать средствами Rational Rose модель системы поддержки заказа и учета товаров в бакалейной лавке.

В бакалейной лавке для каждого товара фиксируется место хранения (определенная полка), количество товара и его поставщик. Система поддержки заказа и учета товаров должна обеспечивать добавление информации о новом товаре, изменение или удаление информации об имеющемся товаре, хранение (добавление, изменение и удаление) информации о поставщиках, включающей в себя название фирмы, ее адрес и телефон. При помощи системы составляются заказы поставщикам. Каждый заказ может содержать несколько позиций, в каждой позиции указываются наименование товара и его количество в заказе. Система учета по требованию пользователя формирует и выдает на печать следующую справочную информацию:

- список всех товаров;
- список товаров, имеющихся в наличии;
- список товаров, количество которых необходимо пополнить;
- список товаров, поставляемых данным поставщиком.

В ходе выполнения этого варианта задания должна быть разработана схема базы данных, хранящей информацию о товарах, заказах и поставщиках.

4.14. Библиотечная система

Требуется разработать средствами Rational Rose модель системы автоматизирующей деятельность библиотеки.

Система поддержки управления библиотекой должна обеспечивать операции (добавление, удаление и изменение) над данными о читателях. В регистрационном списке читателей хранятся следующие сведения: фамилия, имя и отчество читателя; номер его читательского билета и дата выдачи билета. Наряду с регистрационным списком системой должен поддерживаться каталог библиотеки, где хранится информация о книгах: название, список авторов, библиотечный шифр, год и место издания, название издательства, общее количество экземпляров книги в библиотеке и количество экземпляров, доступных в текущий момент. Система обеспечивает добавление, удаление и изменение данных каталога, а также поиск книг в каталоге на основании введенного шифра или названия книги. В системе осуществляется регистрация взятых и возвращенных читателем книг. Про каждую выданную книгу хранится запись о том, кому и когда была выдана книга, и когда она будет

возвращена. При возврате книги в записи делается соответствующая пометка, а сама запись не удаляется из системы. Система должна выдавать следующую справочную информацию:

- какие книги были выданы за данный промежуток времени;
- какие книги были возвращены за данный промежуток времени;
- какие книги находятся у данного читателя;
- имеется ли в наличии некоторая книга.

Вариант задания предусматривает разработку схемы базы данных, хранящей список читателей, каталог книг и записи о выдаче книг.

4.15. Интернет-магазин

Требуется разработать средствами Rational Rose модель программного обеспечения Интернет-магазина.

Интернет-магазин позволяет делать покупки с доставкой на дом. Клиенты магазина при помощи программы-браузера имеют доступ к каталогу продаваемых товаров, поддержку которого осуществляет Интернет-магазин. В каталоге товары распределены по разделам. О каждом товаре доступна полная информация (название, вес, цена, изображение, дата изготовления и срок годности). Для удобства клиентов предусмотрена система поиска товаров в каталоге. Заполнение каталога информацией происходит автоматически в начале рабочего дня, информация берется из системы автоматизации торговли.

При отборе клиентами товаров поддерживается виртуальная «торговая корзина». Любое наименование товара может быть добавлено в «корзину» или изъято в любой момент по желанию покупателя с последующим пересчетом общей стоимости покупки. Текущее содержимое «корзины» постоянно показывается клиенту.

По окончании выбора товаров производится оформление заказа и регистрация покупателя. Клиент указывает в регистрационной форме свою фамилию, имя и отчество, адрес доставки заказа и телефон, по которому с ним можно связаться для подтверждения сделанного заказа. Заказы передаются для обработки в систему автоматизации торговли. Проверка наличия товаров на складе и их резервирование Интернет-магазином не производятся. Дополнительно требуется разработать схему базы данных, хранящей заказы.

При выполнении этого варианта задания рекомендуем ознакомиться с работой [Коналлен-2001]. Следует определиться, по какому архитектурному шаблону будет строиться Web-приложение («тонкий клиент» или «толстый клиент»). В соответствии с выбранным шаблоном следует построить модели клиентской части магазина и серверной части, промоделировать связи между частями приложения. Для Web-приложений типичными являются следующие классы:

- клиентская Web-страница;
- серверная Web-страница (например, CGI-скрипт);
- HTML-форма;
- объект JavaScript.

Дополнительные *связи* между классами Web-приложений:

- link - ссылка с одной страницы на другую;
- build - связь между CGI-скриптом и клиентской страницей, генерируемой при его выполнении;
- submit - связь между формой и серверной Web-страницей, принимающей данные из формы.

Типичные компоненты:

- Web-страница (HTML-файл),
- Active Server Page (ASP),
- Java Server Page (JSP),
- сервлет,
- библиотека скриптов (например, подключаемый файл с Javascript-функциями).

4.16. WWW-конференция

Требуется разработать средствами Rational Rose модель программного обеспечения WWW-конференции.

WWW-конференция представляет собой хранилище сообщений в сети Интернет, доступ к которому осуществляется при помощи браузера. Для каждого сообщения конференции хранятся значения следующих полей: номер сообщения, автор, тема, текст сообщения, дата добавления сообщения, ссылка на родительское сообщение. Начальной страницей конференции является иерархический список сообщений. Верхний уровень иерархии составляют сообщения, открывающие новые темы, а подуровни составляют сообщения, полученные в ответ на сообщения верхнего уровня. Сообщение-ответ всегда имеет ссылку на исходное сообщение. В списке отображаются только темы сообщений, их авторы и даты добавления. Просматривая список, пользователь выбирает сообщение и по гиперссылке открывает страницу с текстом сообщения. Помимо текста на этой странице отображается список (иерархический) сообщений являющихся ответами, ответами на ответы и т.д. Для удобства пользователей необходимо предусмотреть поиск сообщений по автору или по ключевым словам в теме или тексте сообщения.

Сообщения добавляются в конференцию зарегистрированными пользователями, которые при отправке сообщения должны указать своё имя и пароль. Регистрирует новых пользователей модератор конференции - её ведущий. При регистрации пользователь заполняет специальную форму, содержимое которой затем пересыпается модератору и запоминается в базе пользователей. Модератор решает, регистрировать пользователя или нет, и отправляет свой ответ.

При добавлении сообщений пользователь имеет возможность начать новую тему или ответить на ранее добавленные сообщения. После добавления сообщения оно доступно для чтения всем пользователям (даже незарегистрированным), и список сообщений обновляется.

Модератор имеет право по тем или иным причинам удалять сообщения любых авторов. Он также может наказывать пользователей, нарушающих правила поведения в конференции, лишая на некоторое время пользователя возможности добавлять и редактировать сообщения.

Вариант задания включает в себя разработку схемы базы данных для хранения сообщений конференции и информации об её участниках.

Выполняющим это задание полезно ознакомиться с заключительным замечанием к варианту «Интернет-магазин». Наиболее подходящей архитектурой для WWW-конференции является «тонкий клиент», поскольку клиентская часть практически не содержит «бизнес-логики». Единственным её элементом, который может выполняться на стороне клиента, является проверка правильного заполнения полей формы, перед отправкой её содержимого на сервер.

4.17. Каталог ресурсов Интернет

Требуется разработать средствами Rational Rose модель программного обеспечения каталога ресурсов сети Интернет.

В каталоге хранится следующая информация о ресурсах: название ресурса, уникальный локатор ресурса (URL), раздел каталога, в котором содержится ресурс, список ключевых слов, краткое описание, дата последнего обновления, контактная информация.

Доступ пользователей к каталогу осуществляется при помощи браузера. Пользователи каталога могут добавлять новые ресурсы, информация о которых не была внесена ранее. Ресурсы в каталоге классифицируются по разделам. Полный список ресурсов каждого раздела должен быть доступен пользователям. Пользователям каталога должны быть предоставлены возможности по поиску ресурсов. Поиск осуществляется по ключевым словам. Если пользователь не доволен результатами поиска, он может уточнить запрос (осуществить поиск среди результатов предыдущего поиска). Должна быть возможность выдавать результаты поиска в разной форме (вывод всей информации о ресурсах или частичной). Пользователь может отсортировать список ресурсов по релевантности (соответствию ключевым словам из запроса) или по дате обновления.

Поскольку содержание ресурсов Интернет со временем изменяется необходимо следить за датой последнего обновления, периодически опрашивая Web-сайты, URL которых хранятся в каталоге.

Вариант задания включает в себя разработку схемы базы данных для хранения сообщений конференции и информации об её участниках.

Выполняющим это задание полезно ознакомиться с заключительным замечанием к варианту «Интернет-магазин». Как и в варианте «WWW-конференция» самой подходящей архитектурой для

каталога является «тонкий клиент», поскольку клиентская часть практически не включает в себя функций «бизнес-логики» кроме проверки содержимого форм перед пересылкой на сервер.

4.18. Будильник

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного микропроцессора для будильника.

На экране будильника постоянно отображается текущее время (часы и минуты, например: 12:00), двоеточие между числом часов и числом минут зажигается и гаснет с интервалом в полсекунды.

Управление будильником осуществляется следующими кнопками:

- кнопкой режима установки времени,
- кнопкой режима установки времени срабатывания,
- двумя отдельными кнопками для установки часов и минут,
- кнопкой сброса сигнала «СБРОС».

На будильнике имеется переключатель режима работы со следующими положениями: «ВЫКЛ», «ВКЛ», «РАДИО» и «ТАЙМЕР».

Для установки текущего времени нужно нажать на кнопку режима установки и, при нажатой кнопке, нажимать на кнопки установки часов и минут. При каждом нажатии на кнопки, устанавливаемое значение увеличивается на одну единицу (один час или одну минуту соответственно). При достижении максимального значения производится сброс. Для установки времени срабатывания будильника нужно нажать на кнопку режима установки времени срабатывания и, держа кнопку нажатой, нажимать на кнопки установки часов и минут. Когда переключатель режима работы находится в положении «ВКЛ», при достижении времени срабатывания происходит подача звукового сигнала в течение одной минуты. Сигнал можно прервать, нажав на кнопку «СБРОС». При этом сигнал должен быть возобновлен через пять минут. При установке переключателя в положение «ВЫКЛ» звуковой сигнал не подается.

Когда переключатель находится в положении «РАДИО» работает радиоприемник. При переводе переключателя в положение «ТАЙМЕР» включается радиоприемник на тридцать минут, а затем часы переходят в состояние будильника (аналогично положению «ВКЛ»). При нажатии на кнопку режима установки времени, будильник должен отображать время срабатывания.

4.19. Генеалогическое дерево

Требуется разработать средствами Rational Rose модель системы для поддержки генеалогических деревьев.

Система хранит сведения о персонах (Ф.И.О., пол, дата рождения, дата смерти, биография) и о родственных связях между ними. Связи бывают только трех видов: «мужья-жены», «дети-родители» и «братья-сестры». Система обеспечивает возможность добавления данных о новых персонах и родственных связях, изменение введенных данных и удаление ненужных данных. Система следит за непротиворечивостью вводимых данных. Например, недопустимо, чтобы человек был собственным предком или потомком.

Разработанная модель должна содержать схему базы данных для хранения генеалогических деревьев.

Пользователи системы могут осуществлять поиск полезной информации по дереву:

- находить для указанного члена семьи его детей;
- находить для указанного члена семьи его родителей;
- находить для указанной персоны братьев и сестер, если таковые есть;
- получать список всех предков персоны;
- получать список всех потомков персоны;
- получать список всех родственников персоны;
- прослеживать цепочку родственных связей от одной персоны до другой (например, если Петр является шурином Ивана, то на запрос о родственных связях между Петром и Иваном выдается такой результат: «Петр - брат Ольги, Ольга - жена Ивана»).

4.20. Телевизор

Требуется разработать средствами Rational Rose модель встроенной системы управления работой телевизора.

В телевизоре имеются следующие устройства: приемник телевизионного сигнала, устройство отображения картинки, память каналов, память настроек, управляющие кнопки, пульт дистанционного управления (ДУ). Управление телевизором осуществляется при помощи кнопок на корпусе (их четыре: «ВКЛ / ВЫКЛ», « - », « + », кнопка начальной установки) и пульта ДУ. Кнопка «ВКЛ / ВЫКЛ» позволяет включать и выключать телевизор. После включения телевизора на экран отображается передача, идущая по каналу №1, при этом используются параметры изображения и значение громкости, сохраненные в памяти настроек.

Память каналов телевизора хранит до 60 каналов. Каналы нумеруются, начиная с нуля. Последовательное переключение каналов осуществляется при помощи кнопок «-» и «+». Нажатие на «+» переключает телевизор на канал с номером, на единицу большим (с 59-го канала телевизор переключается на 0-ой). Нажатие на «-» переключает телевизор на канал с номером, на единицу меньшим (с 0-го канала телевизор переключается на 59-ый).

При нажатии на кнопку начальной установки очищается память каналов телевизора, после чего осуществляется поиск передач и сохранение их частот в памяти каналов. Поиск начинается с нижней границы рабочего диапазона телевизора. На экран телевизора выводится «синий экран». Рабочая частота постепенно увеличивается до тех пор, пока приемник не обнаружит телевизионный сигнал. Найденная передача выводится на экран в течение 10 секунд. Также отображается номер, под которым найденный канал будет сохранен в памяти (начиная с 1).

Затем поиск продолжается до тех пор, пока не достигнута верхняя граница диапазона, или пока не заполнена вся память каналов.

Телевизор принимает управляющие сигналы с пульта ДУ. На пульте ДУ расположены следующие кнопки:

- кнопки с цифрами «0»...«9» для прямого переключения канала (по номеру);
- кнопки «П-» и «П+» для последовательного переключения каналов;
- кнопки «Г-» и «Г+» для изменения громкости;
- кнопки «МЕНЮ», «<» и «>» для доступа к экранному меню. Для прямого переключения на нужный канал его номер набирается

с помощью кнопок с цифрами. После нажатия первой цифры в течение 5 секунд ожидается нажатие второй. Если вторая цифра не была нажата, то номер канала считается состоящим из одной цифры и осуществляется переключение на него.

Кнопки «П-» и «П+» на пульте имеют те же функции, что и кнопки «-» и «+» на корпусе телевизора. Кнопки «Г-» и «Г+» позволяют увеличивать или уменьшать громкость. Каждое нажатие на «Г-» уменьшает громкость на одну единицу, пока она больше нуля, «Г+» увеличивает громкость на единицу, пока не достигнуто максимальное значение. Текущее значение громкости сохраняется в памяти настроек.

Кнопки «МЕНЮ», «<» и «>» позволяют устанавливать значения настроек, хранящихся в памяти телевизора. При нажатии на кнопку «МЕНЮ» внизу экрана возникает надпись «ЯРКОСТЬ» и полоса, отображающая текущее значение яркости. Кнопками «<» и «>» яркость можно уменьшить или увеличить. При работе с меню нажатия на все остальные кнопки игнорируются. После повторного нажатия на кнопку «МЕНЮ» значение яркости запоминается в памяти настроек, и осуществляется переход к настройке контрастности. Настройка контрастности и остальных параметров (четкости, цветовой гаммы) происходит аналогично. Нажатие на кнопку «МЕНЮ» по окончании настройки цветовой гаммы (последнего пункта меню) приводит к окончанию работы с меню. Выход из меню также осуществляется в том случае, если в течение 15 секунд не была нажата ни одна кнопка.

4.21. Система поддержки составления расписания занятий

Требуется разработать средствами Rational Rose модель системы поддержки составления расписания занятий.

Система обеспечивает составление расписания некоторого учебного заведения, внесение в расписание изменений, выдачу полного расписания и дополнительной информации (например, по итоговому расписанию составляется расписание указанной группы на заданный день или неделю).

В расписании фиксируются время и место проведения занятия, предмет и преподаватель, проводящий занятие, а также номер группы, для которой это занятие проводится. Расписание не должно содержать коллизий (например, разные занятия не должны пересекаться друг с другом по месту и времени их проведения, один преподаватель не может вести одновременно два разных занятия, в одно и то же время у одной и той же группы не может быть два различных занятия и т. д.).

При работе над этим вариантом задания необходимо разработать схему базы данных для хранения расписания.

4.22. Домофон

Требуется разработать средствами Rational Rose модель программного обеспечения встроенного микропроцессора домофона.

Домофон регулирует доступ в подъезд многоквартирного дома. В подъезде имеется дверь с замком. С наружной стороны двери установлена внешняя панель домофона, на которой находятся кнопки для связи с каждой квартирой, микрофон и динамик. В каждой квартире находится внутренняя панель домофона с кнопками: «СВЯЗЬ», «БЛОКИРОВКА» и «ОТКРЫТЬ». Кроме того, на внутренней панели имеется микрофон и динамик.

Жильцы могут открывать дверь ключом. Посетитель может нажать кнопку квартиры на внешней панели. При этом в квартире раздается звонок (если подача звонка в квартиру не заблокирована). Услышав звонок, жилец квартиры нажимает на кнопку «СВЯЗЬ» внутренней панели домофона, после чего домофон устанавливает звуковое сообщение между жильцом и посетителем. Звуки, произносимые посетителем в микрофон, установленный на внешней панели, воспроизводятся в динамике, установленном в квартире. Звуки из микрофона в квартире, передаются в динамик на внешней панели. После сеанса связи жилец может нажать на кнопку «ОТКРЫТЬ», чтобы замок на двери в подъезд открылся, и посетитель смог войти. По истечении минуты замок должен снова заблокировать вход в подъезд.

Жилец, который желает, чтобы его не беспокоили, может отключить подачу звонка в свою квартиру, нажав на кнопку «БЛОКИРОВКА». Повторное нажатие на эту кнопку вновь включает подачу звонка.