Министерство образования республики беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информатики

А.В. Галковский

Применение языка UML при объектно-ориентированном проектировании

Методическое пособие
для студентов специальности 31 03 04 “Информатика”
дневной формы обучения

Минск 2004

УДК 004.436.4 (075.8)
ББК 32.973-02 я 73
Г 16

Рецензент:
доцент кафедры экономической
информатики, канд. техн. наук В.Н. Комличенко

Галковский А.В.

Г 16 Применение языка UML при объектно-ориентированном проектировании: Метод. пособие для студ. спец. 31 03 04 «Информатика» дневной формы обучения/А.В. Галковский. – Мн.: БГУИР, 2004. – 36 с.
ISBN 985-444-580-1

Методическое пособие является дополнительным учебным материалом для студентов, изучающих обьектно-ориентированное программирование и проектирование программного обеспечения с использованием системы Rational Rose.

УДК 004.436.4 (075.8)
ББК 32.973-02 я 73

ISBN 985-444-580-1

СОДЕРЖАНИЕ

Введение

В настоящее время для анализа и проектирования объектно-ориентированных программных систем все более широкое применение получают языки визуального моделирования и средства автоматизации проектирования программного обеспечения.

Языки визуального моделирования появились сравнительно недавно, в конце 80-х годов. В настоящее время различают три поколения. К третьему поколению относится язык UML (Unified Modeling Language - Унифицированный язык моделирования). Был разработан в 1994-1997 гг., основные разработчики — Г.Буч, Дж. Рамбо, И. Джекобсон. В 1997 г. OMG объявила UML стандартным языком объектно-ориентированного моделирования. OMG (Object Management Group) – международный консорциум, основанный в 1989 г., в который в настоящее время входят более 800 авторитетных компаний; одной из основных его целей является разработка стандартов и спецификаций в области объектно-ориентированных технологий.

UML может использоваться для визуализации, спецификации, конструирования и документирования результатов объектно-ориентированных программных проектов. UML — это не визуальный язык программирования, но его модели прямо транслируются в текст на языках программирования (Java, C++, Visual Basic, Ada 95, Object Pascal) и даже в таблицы для реляционной БД.

Применение языков визуального проектирования предполагает использование некоторых средств компьютерной проддержки процесса проектирования. Речь идет о CASE-системах (CASE - Computer Aided Software Engineering - система автоматизированного проектирования программного обеспечения). Это инструмент, который позволяет существенно сократить сроки и затраты, оказывая помощь инженеру в проведении рутинных операций, облегчая его работу на самых разных этапах жизненного цикла разработки программного обеспечения. Наиболее известной объектно-ориентированной CASE-системой является Rational Rose, которая основана на применении UML в качестве базового средства описания проекта.

В данном пособии будут рассматриваться основы языка UML, а также применение системы Rational Rose при объектно-ориентированном проектировании ПО.

1. Обзор UML

UML - это язык визуального моделирования для решения задач общего характера, который используется при определении, визуализации, конструировании и документировании программной системы. С помощью языка UML можно фиксировать решения, принятые при создании различных систем. Он используется для того, чтобы лучше понимать, проектировать, поддерживать и контролировать эти системы.

UML позволяет отображать и статическую структуру, и динамическое поведение системы. Система моделируется как группа дискретных объектов, которые взаимодействуют друг с другом таким образом. В статической структуре задаются типы объектов, значимые для системы и ее реализации, а также отношения между этими объектами. Динамическое поведение определяет историю объектов и их взаимодействие для достижения конечной цели.

UML не является языком программирования. Программный код можно получить на основе созданной модели с помощью инструментальных средств, поддерживающих UML и содержащих генераторы кода. С другой стороны, на основе исходного кода можно восстановить UML-модели уже существующих программ.

2. Представления UML

Язык UML позволяет представлять проектируемую систему с различных точек зрения. Различают три основных представления: структурная классификация, динамическое поведение и управление моделью.

Структурная классификация описывает системные сущности и их отношения между собой. В число классификаторов, имеющихся в моделях UML, входят классы, варианты использования, компоненты и узлы. Классификаторы являются базой, на которой строится динамическое поведение системы.

Динамическое поведение описывает поведение системы во времени. Поведение можно определить как ряд изменений в мгновенных снимках системы, полученных со статической точки зрения.

Представление управления моделью — это описание разбиения модели на иерархические блоки. Групповой организационный блок называется пакетом. Отдельные пакеты включают модели и подсистемы. Представление управления моделью организует все остальные представления моделей и позволяет осуществлять процесс разработки и управления конфигурацией.

Различные представления проекта описываются с помощью диаграмм.

- диаграмма вариантов использования (use case diagram);
- диаграмма классов (class diagram);
- диаграмма объектов (object diagram);
- диаграмма последовательности (sequence diagram);
- диаграмма кооперации (collaboration diagram) ;
- диаграмма состояний (statechart diagram);
- диаграмма деятельности (activity diagram);
- диаграмма компонентов (component diagram);
- диаграмма развертывания (deployment diagram).

2.1. Использование диаграмм при проектировании

С помощью диаграмм UML разработчики создают различные модели системы. Разнообразие моделей позволяет разработчикам отразить различные аспекты системы:

- Варианты использования и их диаграммы применяются для визуализации функциональных возможностей системы.

- Диаграммы взаимодействия показывают, как объекты работают совместно, обеспечивая требуемые функциональные возможности.

- Диаграммы классов и объектов используются для отображения объектов системы и их отношений.

- Диаграммы компонентов иллюстрируют, как классы соотносятся с готовыми физическими компонентами системы.

- Диаграммы размещения применяют для визуализации проекта распределенных систем.

Модель детально описывает, что система содержит и как функционирует, поэтому разработчики могут использовать ее в качестве эскиза или чертежа создаваемой системы. Однако модели используют не только разработчики:

- С помощью диаграмм вариантов использования потребители и менеджеры проекта получат общее представление о системе и смогут принять решение о сфере ее применения.

- С помощью диаграмм вариантов использования и документации менеджеры проекта смогут разделить проект на отдельные управляемые задачи.

- Из документации по вариантам использования аналитики и потребители смогут понять, что будет делать готовая система.

- Изучив ту же документацию, технические писатели смогут приступить к написанию руководства для пользователей и к подготовке планов по их обучению.

- Из диаграмм последовательности и кооперативных диаграмм аналитики и разработчики уяснят, насколько логично работает система, поймут ее объекты и сообщения между ними.

- С помощью документации по вариантам использования, а также диаграмм последовательности и кооперативных диаграмм специалисты по контролю качества смогут получить информацию, необходимую им для написания тестовых сценариев.

- С помощью диаграмм классов и состояний разработчики получат представление о фрагментах системы и их взаимодействии друг с другом.

- Из диаграмм компонентов и размещения эксплуатационный персонал сможет узнать, какие .EXE и .DLL файлы и другие компоненты будут созданы, а также где в сети они должны быть размещены.

2.2. Диаграмма вариантов использования

Диаграмма вариантов использования описывает функционирование системы с точки зрения внешнего наблюдателя. При этом описывается, **что** система делает, но не **как** она этот делает.

Диаграмма представляет собой совокупность взаимосвязанных сценариев. Сценарий – это пример того, что происходит, когда кто-нибудь взаимодействует с системой. Пример сценария для медицинского центра: “Пациент звонит в медицинский центр, чтобы получить назначение к врачу на ближайшее время. Диспетчер в медицинском центре находит ближайший свободный промежуток времени в журнале учета и делает соответствующую запись на прием к врачу”. Описание этой ситуации с использованием языка UML показано на рис. 1.

Рис.1. Пример диаграммы вариантов использования

В этом представлении описывается функционирование системы с точки зрения ее пользователя, который называется в моделировании действующим лицом (actor). Действующие лица представляют собой роли людей или объектов в системе. Они инициируют события для определенных задач. Вариант использования (use case) — это функциональный блок, описывающий некоторую функцию проектируемой системы. На диаграмме представляется в виде овала с названием функции. В нашем примере это функция “Сделать назначение” (make appointment). Линия, соединяющая действующее лицо и вариант использования, называется ассоциацией связи (communication association).

Диаграмма использования – это совокупность действующих лиц, вариантов использования и связей между ними. Задача проектировщика - выявить все действующие лица системы и все варианты ее использования, а также указать, какие действующие лица в каких вариантах использования фигурируют.

Более подробная диаграмма для описания медицинского центра приведена на рис. 2.

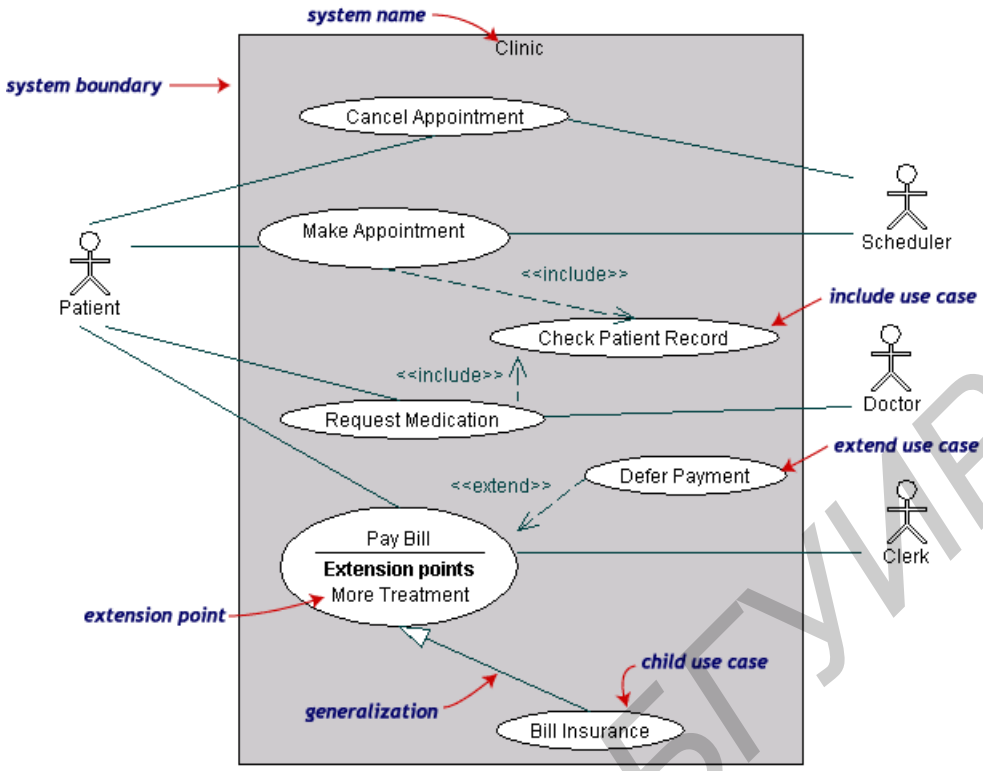

Рис.2. Диаграмма вариантов использования медицинской системы

2.3. Диаграмма классов

В диаграмме классов основной акцент сделан на описании классов и их взаимоотношений: ассоциаций, обобщений и различных видов зависимостей, например реализаций и использования. Класс (class) представляет собой отдельную концепцию моделируемой системы. Классом может быть нечто, относящееся к материальному миру (например самолет), деловым отношениям (заказ), логике (расписание занятий) или поведению (задача). Класс описывает множество объектов со сходной структурой, поведением и отношениями. Классы — это основное понятие, вокруг которого строится объектно-ориентированная система.

Объекты, которые определяются классом, обладают состоянием и поведением. Состояние описывается атрибутами и ассоциациями. Атрибуты обычно используются для хранения значений, не обладающих индивидуальностью (например чисел и строк). Ассоциации нужны для связывания объектов, обладающих индивидуальностью. Отдельные элементы поведения описываются операциями класса. Реализацией операции является метод. История жизни объекта описывается конечным автоматом, который привязан к классу.

Диаграмма классов, которая представлена на рис.3, моделирует заказ клиента из розничного каталога. Основное понятие в этой схеме – «заказ» (класс Order) . С «заказом» связаны еще два понятия: «покупатель» (Customer) и «оплата» (Payment). Возможные виды оплаты: за наличные деньги (класс Cash), чек (Check) или кредит (Credit). Частью класса Order является класс OrderDetails, который хранит список пунктов (Item) заказа .

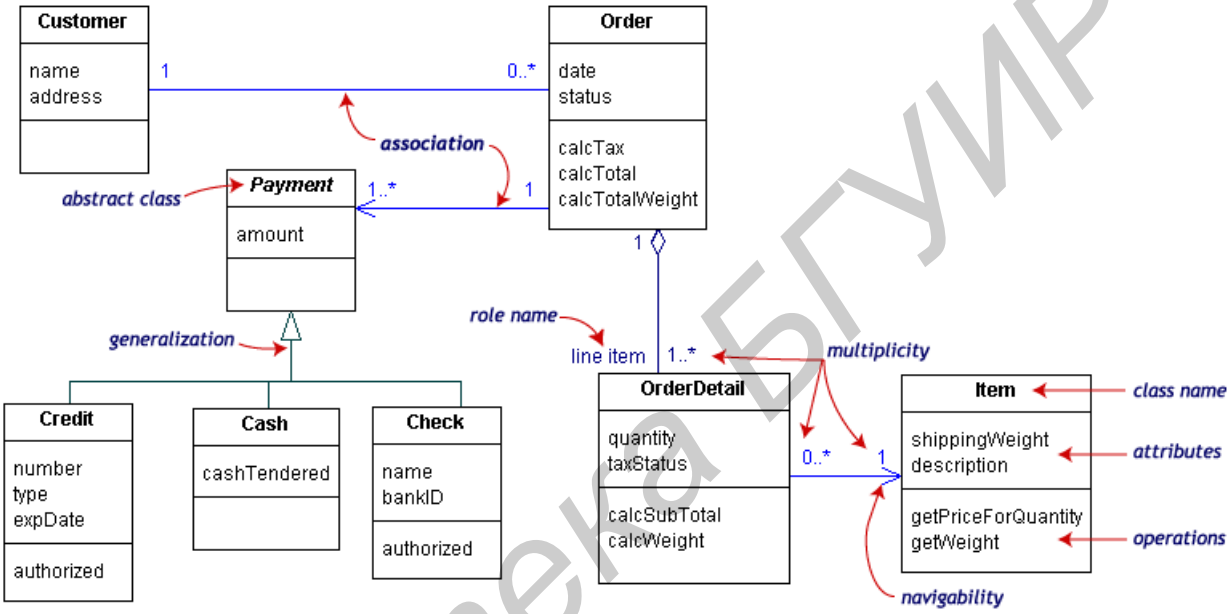

Рис.3. Диаграмма классов “Заказ товаров”

Графически класс изображается в виде прямоугольника. Атрибуты и операции класса перечисляются в горизонтальных отделениях этого прямоугольника. Отношения между классами выражаются при помощи различных линий и дополнительных обозначений, которые ставятся либо над самими линиями, либо у их концов.

На диаграмме классов имеется три вида отношений: ассоциация (association), агрегация (aggregation) и обобщение (generalization).

2.3.1. Информация о классе

Нотация класса в UML представляет собой прямоугольник, разделенный на три части, в которых задается имя класса, имена его атрибутов и операций.

Атрибуты и операции могут быть помечены в соответствии с типом доступа и видимости. Например, на рис.4 приводится развернутое описание класса Order.

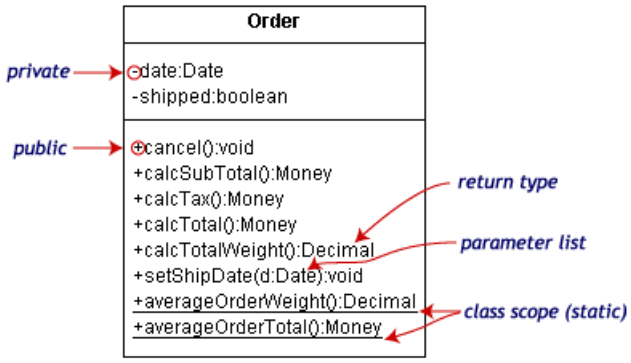

Рис.4. Описание класса

При описании класса используются следующие соглашения UML:

- Статические (static) члены класса подчёркиваются.
- Описание операций имеет следующую форму:

<спецификатор_доступа> <имя_операции> (*<список_параметров>*) : *<тип_возвращаемого_значения >*

Спецификатор доступа (access specifier) может принимать одно из следующих значений (табл.1).

Таблица 1

Значения спецификаторов доступа

Символ	**Тип доступа**
+	Public
-	Private
#	Protected

Список параметров (parameter list) - это последовательность параметров операции, разделенных запятыми. Форма каждого параметра:

<имя_параметра> : <тип_параметра>

Пример описания операции:

#MyOperation(param1 : integer, param2 : integer) : long

т.е. MyOperation() является защищенным (# - protected) членом класса, имеет два параметра целого типа (integer), тип возвращаемого значения – длинное целое (long).

2.3.2. Отношение обобщения

Обобщение (generalization) — это вид отношений между общим описанием и специфическим описанием, которое основывается на общем и детализирует его. Специфическое описание полностью согласуется с общим (имеет те же свойства, члены и отношения), но обладает также и некоторой дополнительной информацией. В применении к классам существуют другие названия: суперкласс и подкласс.

Обобщение можно делать в том случае, когда поведение и состояние различных классов имеют общие черты. Обобщение связывает более конкретные классы (подклассы) с более общими (суперклассами). Суперклассы обладают свойствами, общими для нескольких подклассов.

Еще одной целью обобщения является дать возможность разработчику описывать класс, который содержит общие свойства классов - потомков. Это называется наследованием.

На диаграммах обобщение изображается в виде стрелки с большим полым треугольником, идущей от прямого потомка к прямому предку. Несколько стрелок-обобщений можно объединить в одно дерево, так что стрелка будет указывать на прямого предка, а несколько ветвей — на прямых потомков.

На диаграмме: класс Payment является суперклассом для классов Cash, Check и Credit.

Исходный код на языке C++ , который является реализацией данной части диаграммы, приведен ниже :

class Payment { ... };

class Cash : public Payment { ... };

class Check : public Payment { ... };

class Credit : public Payment { ... };

2.3.3. Интерфейсы

Интерфейс представляет собой набор описаний операций, то есть он содержит описание поведения объектов, в котором не указываются их реализация и состояние. Интерфейс описывает операции, но не атрибуты и не имеет исходящих ассоциаций. Интерфейс можно рассматривать как абстракцию для описания некоторого поведения, которое реализуется в производных классах.

Диаграмма, представленная на рис.5, описывает модель конференции. Класс **SessionTalk** – это модель отдельной презентации, а класс **Session** – это набор презентаций для одного дня.

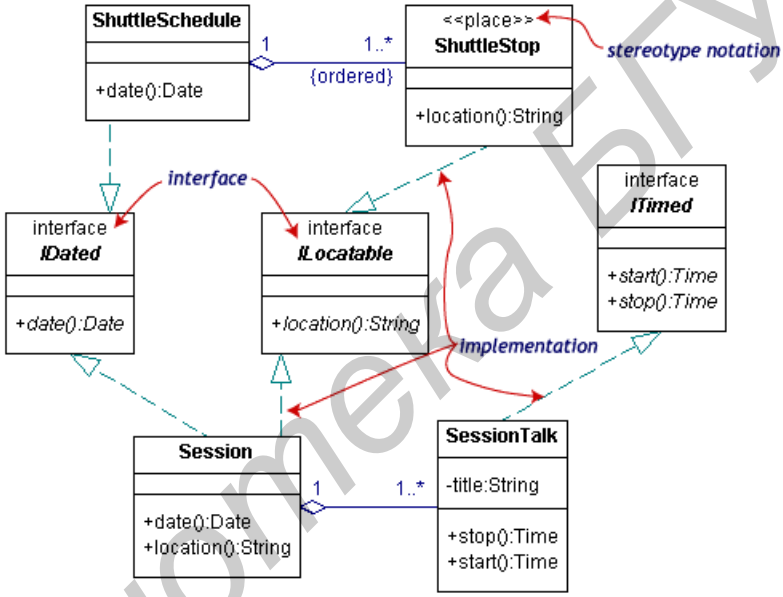

Рис.5. Интерфейсы на диаграмме классов

Имеется три интерфейса: ***IDated***, ***ILocatable***, и ***ITimed***. Имена интерфейсов обычно начинаются с символа ***I*** и имеют шрифт типа *italic* (шрифт с наклоном).

В качестве примера можно рассмотреть часть диаграммы: класс **Session** и интерфейсы ***IDated***, ***ILocatable***. Эти два интерфейса определяют функциональность «хранить дату» и «хранить расположение». Это делается за счет определения в интерфейсах методов date() и location() соответственно. Класс **Session** является реализацией (implementation) обоих этих интерфейсов, поскольку такое понятие, как «сессия», подразумевает функциональность «должна проводиться в определенное время и в определенном месте». Необходимо обратить

внимание на то, что в классе происходит реализация как метода date(), так и locate(). Графически отношение реализации похоже на обобщение, но отличается тем, что линия связи не сплошная, а прерывистая.

Существует две возможные нотации для интерфейсов (рис.6). Выше была показана первая нотация, согласно которой интерфейсы изображаются в виде прямоугольников.

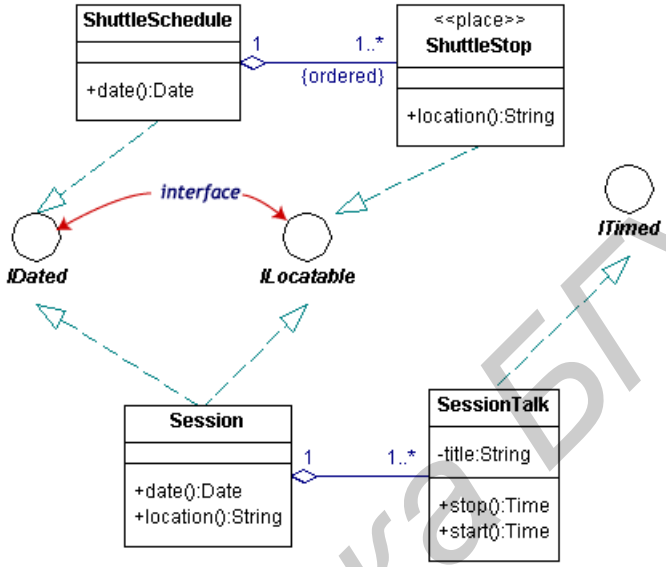

Рис.6. Альтернативное изображение интерфейсов на диаграмме

Во второй нотации интерфейсы представлены в виде кругов с линиями, соединяющими их с реализующими классами. Эта нотация более компактна и упрощает диаграмму, но при этом исчезают некоторые детали описания.

В C++ интерфейс может быть реализован как абстрактный класс с только чисто виртуальными функциями (pure virtual).

```
class IDated
{
 public:
  virtual Date date() = 0;
};
class ILocatable
{
 public:
  virtual String location() = 0;
};
class Session : public IDated, public ILocatable
{
```

```
 public:
  virtual Date date()  {...}
  virtual String location() {...}
};
```

В языке Java имеется соответствующая конструкция языка для реализации интерфейса.

```
public interface IDated
{
  public Date date() ;
};
public interface ILocatable
{
  public  String location();
};
public class Session implements IDated, ILocatable
{
 ...
  public  Date date()  {...}
 public String location() {...}
};
```

2.3.4. Отношение ассоциации

Ассоциация — это отношение между экземплярами двух классов. Необходимо устанавливать отношение ассоциации между двумя классами в том случае, если экземпляр одного класса должен знать об экземпляре другого, для того чтобы выполнять свою работу.

Каждая связь ассоциации с классом называется полюсом ассоциации. Вся основная информация об ассоциации прикрепляется к одному из ее полюсов. Полюса ассоциации обладают именами ролей (role name) и видимостью (navigability). Наиболее ценное их свойство — множественность (multiplicity), которая указывает на то, сколько экземпляров одного класса может быть связано с одним экземпляром другого класса. Множественность обозначается числом или числовым интервалом. На диаграмме классов, приведенной на рис.3, видно, что может быть только один экземпляр Customer для каждого Order, то есть заказу соответствует только один клиент. Но, с другой стороны, у клиента может быть много заказов.

В табл. 2 представлены наиболее распространенные виды множественности.

Таблица 2

Варианты определения множественности для ассоциации

Множественность	**Описание**
0..1	Отсутствие либо один экземпляр
0..* или *	Неограниченное число экземпляров (включая их отсутствие)
1	Только один экземпляр
1..*	По меньшей мере один экземпляр

На диаграмме ассоциация – это сплошная линия, соединяющая два класса. Имя ассоциации пишется у этой линии, а имя роли и множественность обозначаются у полюсов.

На некоторых линиях ассоциации присутствует стрелка. Это стрелка видимости (navigability), которая показывает направление посылки запросов в ассоциации. Стрелка видимости также показывает, какой из классов содержит компоненты для реализации отношения ассоциации, иными словами, кто содержит ссылку на другой объект. Ассоциация без стрелки является двунаправленой.

Например, объекты типа Order могут посылать запросы (сообщения) к объектам типа Payment, но не наоборот.

```
class Order
{
  ...
  protected:
   Payment*   m_apPayments[];
 ...
};
class Payment
{
  ...
  // не имеет ссылок на объекты класса Order
};
```

2.3.5. Отношения агрегации и композиции

Агрегация (aggregation) – так называется ассоциация, представляющая связь «часть - целое». Схематически агрегация изображается в виде незакрашенного ромбика у класса-агрегата.

Композиция (composition) — это более сильная форма агрегации, при которой часть принадлежит только одному целому. При композиции агрегат несет полную ответственность за создание и уничтожение своих частей. Графически композиция изображается в виде закрашенного ромба у класса-агрегата (рис.7).

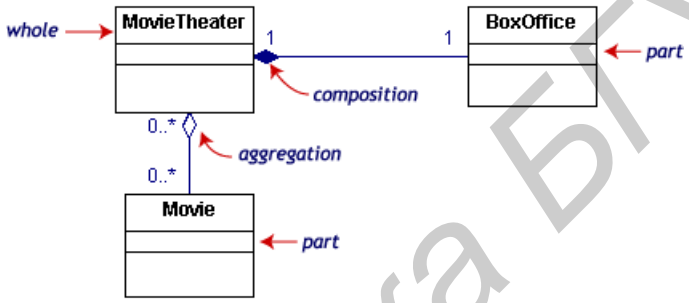

Рис.7. Композиция и агрегация на диаграмме классов

На этой диаграмме показан класс **BoxOffice**, который является частью целого - **MovieTheater**. Между этими классами установлено отношение композиции. Отношение между классами **MovieTheater** и **Movie** – агрегация**.**

Код на языке C++ :

```
class MovieTheater
{
  ...
 protected:
   BoxOffice   m_oBoxOffice;
   Movie *     m_apMovies[];
 ...
};
class BoxOffice
{
 ...
 protected:
   MovieTheater*  m_pOwner;
 ...
};
```

```
class Movie
{
   ...
  protected:
     MovieTheater *   m_apMovieTheaters[];
  ...
};
```

2.4. Диаграммы пакетов и объектов

Для упрощения сложных диаграмм классов можно группировать классы в пакеты. Пакеты представляют собой наборы взаимосвязанных UML-элементов.

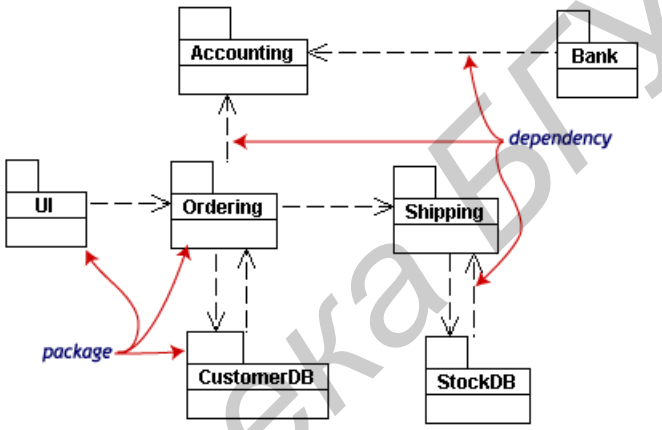

Рис.8. Диаграмма пакетов

Диаграмма, показанная на рис.8, представляет собой бизнес-модель, в которой классы сгруппированы в пакеты. Пакет представлен в виде прямоугольника с изображением «закладки» над ним. Внутри прямоугольника записывается имя пакета. Зависимости (dependencies) между пакетами описываются в виде прерывистых линий со стрелками. Один пакет считается зависимым от другого, если изменения во втором пакете могут привести к изменениям в первом.

Диаграмма объектов показывает взаимосвязи экземпляров некоторых классов. Она используется для пояснения некоторых частей системы со сложными отношениями между объектами, особенно в случае использования рекурсивных отношений.

На диаграмме рис.9 показана модель университетского подразделения (класс **Department**), которое может включать в себя другие подразделения.

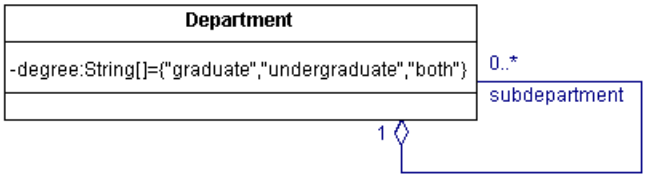

Рис.9. Класс Department с рекурсивной вложенностью

Для пояснения структуры объектов данного класса может быть использована диаграмма объектов (рис.10).

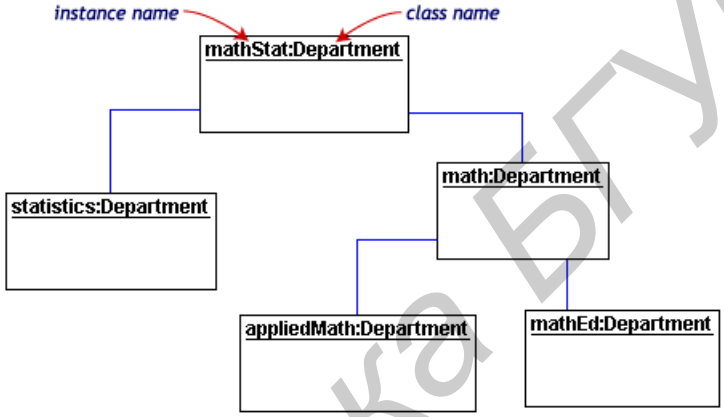

Рис.10. Пример диаграммы объектов

Каждый прямоугольник на диаграмме соответствует одному объекту заданного типа. Имена объектов подчеркнуты, они являются составными: в левой части, до двоеточия, находится собственно имя объекта, а в правой – имя класса, к которому принадлежит объект.

2.5. Диаграмма последовательности

Диаграмма последовательности (sequence diagram) отображает последовательность сообщений между объектами, при этом в деталях показывается, что за сообщение посылается и какому объекту. Каждое сообщение на диаграмме соответствует вызову операции класса. В отличие от диаграммы классов и диаграммы объектов, которые являются статическими, диаграмма последовательности является динамической. Она описывает, как объекты взаимодействуют.

Диаграммы последовательности организованы таким образом, чтобы показывать распространение событий во времени. Временная ось направлена сверху вниз. Объекты, которые принимают участие во взаимодействии, располагаются слева направо, в соответствии с местом, которое они занимают в последовательности сообщений.

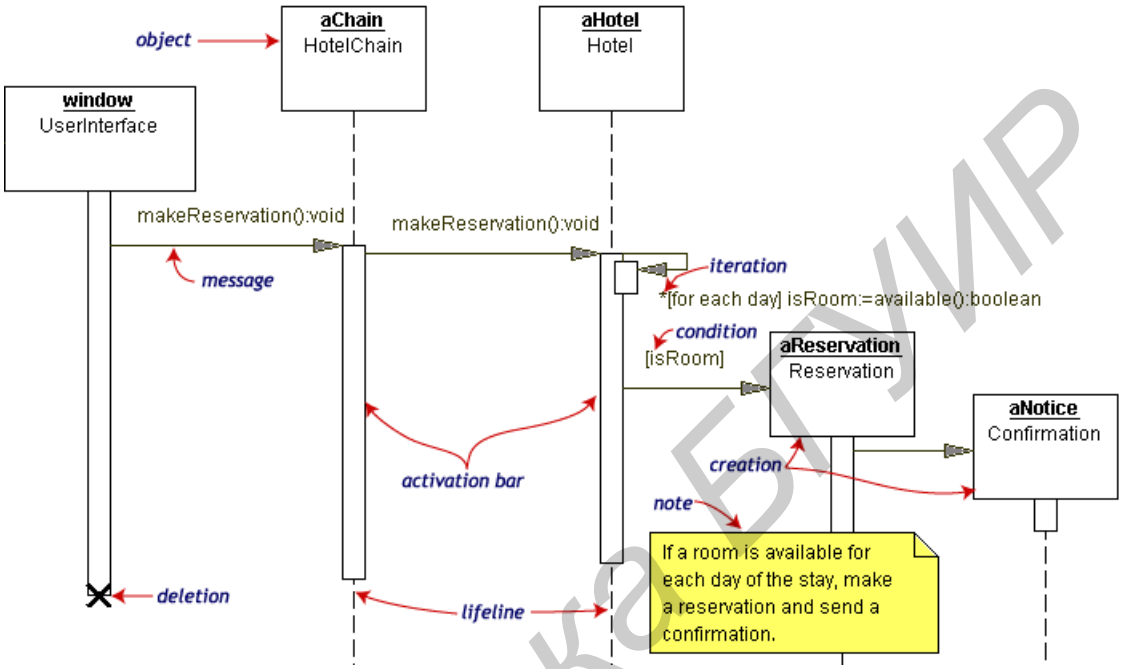

Рис.11. Диаграмма последовательности заказа номера в гостинице

Объекты представлены в виде прямоугольников. Внутри задается имя объекта, а под именем – тип объекта (то есть имя класса). Каждая прерывистая вертикальная линия является линией жизни объекта (lifeline). Линия жизни представляет время существования объекта. Каждая линия со стрелкой соответствует посылке сообщения. Эта линия направлена от объекта-источника к вершине полосы активации (activation bar) на линии жизни объекта-получателя сообщения. Полоса активации представляет продолжительность выполнения сообщения объектом-получателем.

На рис.11 показана диаграмма последовательности, которая описывает процесс резервирования места в гостинице. Инициатором последовательности сообщений является объект с именем «**window**» («окно»). То есть в нашей системе, моделирующей гостиницу, процесс резервирования комнаты начинается с посылки сообщения из оконного интерфейса. Объект «**window**» посылает сообщение *makeReservation()* («зарезервировать») объекту «**aChain**». Тот, в свою очередь, посылает сообщение *makeReservation()* объекту «**aHotel**». Объект

класса **Hotel** посылает сообщение available() самому себе, то есть вызывает один из своих методов для определения доступности комнаты. При этом выполняется цикл (iteration) для проверки доступности комнаты для каждого дня пребывания. В случае успешного выполнения этой операции происходит резервирование и подтверждение заказа. На диаграмме это показано при создании (creation) объектов типа **Reservation** и **Confirmation**. Выражение в квадратных скобках [] описывает условие (condition).

2.6. Диаграмма кооперации

Диаграмма кооперации также является динамической. На диаграмме кооперации (рис.12) представлены объекты и связи между ними.

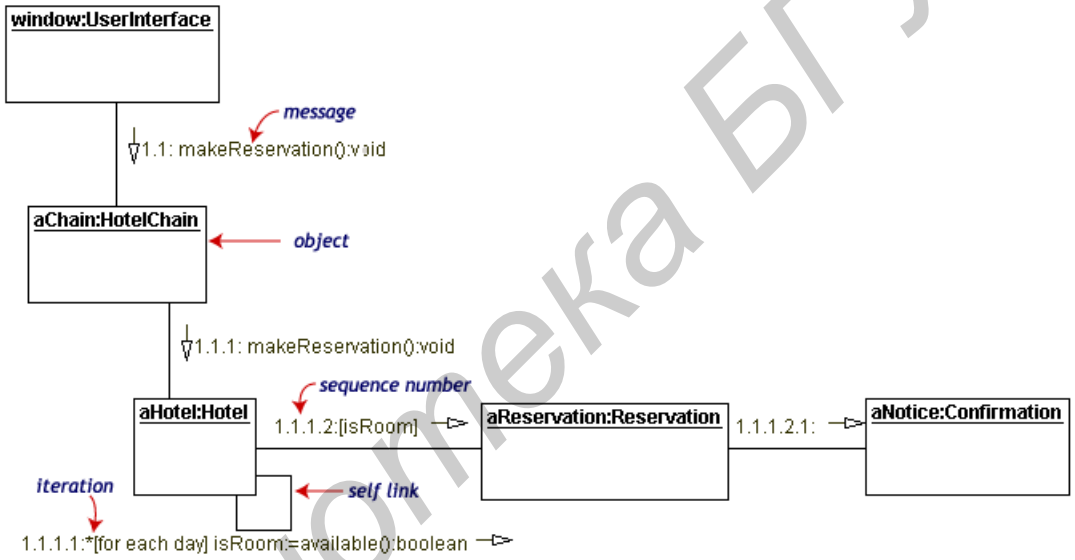

Рис.12. Диаграмма кооперации для описания заказа номера в гостинице

Стрелки, расположенные у линий, изображающих отношения, показывают список сообщений между объектами. Последовательность сообщений задается их нумерацией (sequence number). Диаграмма кооперации и диаграмма последовательности служат для моделирования взаимодействия объектов в системе, однако каждая диаграмма имеет свою специфику. Диаграмма последовательности заостряет внимание на временной последовательности обмена сообщениями. В диаграмме кооперации, напротив, главный акцент сделан на отображении отношений между ролями и связанных с ними сообщений.

2.7. Диаграмма состояний

Диаграмма состояний - это представление в виде конечного автомата. Описывает возможные жизненные циклы объекта и состоит из состояний, соединенных переходами. Каждое состояние — это такой период жизненного цикла объекта, который удовлетворяет определенным условиям. Некоторое событие может привести к переходу, в результате которого объект окажется в новом состоянии. При переходе может выполняться действие, предписанное данному переходу.

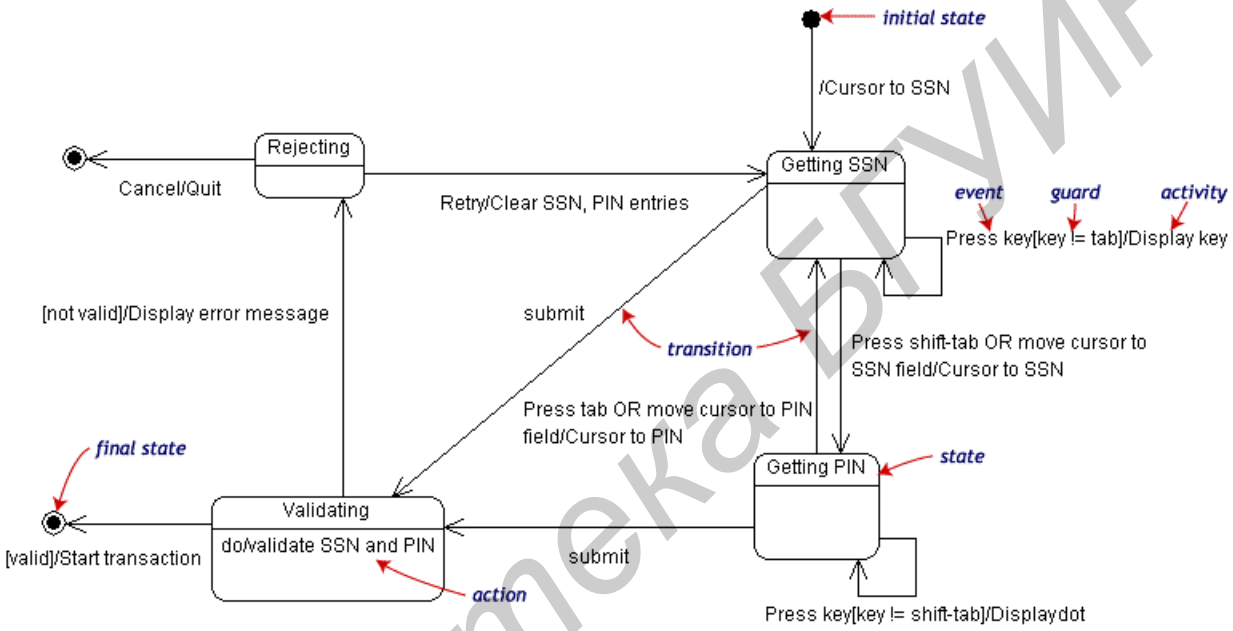

Рис.13. Диаграмма состояний

Пример диаграммы (рис.13) является описанием жизненного цикла объекта, который отвечает за функцию получения доступа в банковскую систему. Для получения доступа необходимо ввести правильный код SSN (social security number) и PIN-код (personal id number), после чего послать информацию на проверку.

Объект может иметь одно из четырех состояний (state): «получение SSN» (**Getting SSN**), «получение PIN» (**Getting PIN**), «проверка» (**Validating**) и «отказ в доступе» (**Rejecting**). Состояния изображены в виде прямоугольников. Переходы (transitions) представлены в виде стрелок, направленных из одного состояния в другое. События (event), которые активизируют переходы, описаны около стрелок. Стартовое состояние автомата описывается в виде черного круга, завершающее состояние – в виде черно-белого круга.

2.8. Диаграмма деятельности

Диаграмма деятельности – это представление деятельности, являющееся разновидностью конечного автомата, в котором показаны длительные вычислительные процессы. Деятельность представляет собой поток работ или выполнение операций. Представление деятельности отображает как последовательные, так и параллельные виды деятельности. Диаграмма деятельности и диаграмма состояний являются родственными диаграммами. Но если диаграмма состояний рассматривает объект как некоторый процесс, то диаграмма деятельности фокусирует внимание на потоках деятельности, включенных в единый процесс. Диаграмма деятельности (рис.14) показывает, как эти потоки и операции зависят друг от друга.

Например, мы используем следующий процесс: "Получить деньги из банка через банковский автомат".

В нашей системе присутствуют три класса объектов, обеспечивающих некоторую деятельность. Это пользователь (**Customer**), банковский автомат (**ATM**) и банк (**Bank**). Процесс начинается с черного круга вверху и заканчивается в черно-белом круге внизу диаграммы.

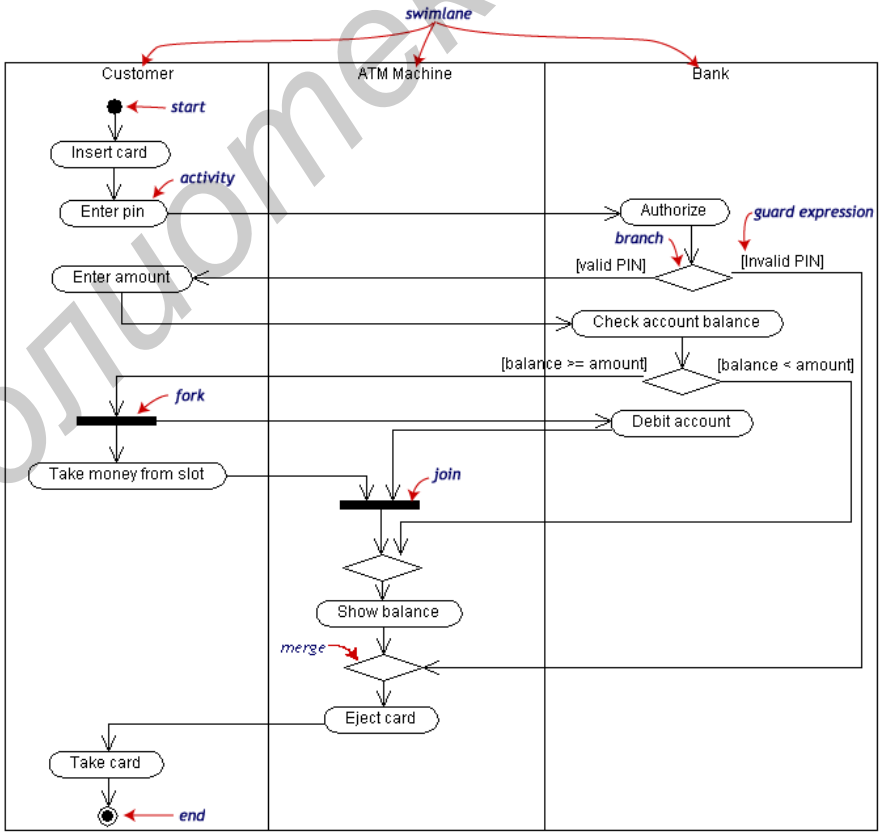

Рис.14. Диаграмма деятельности

Диаграмма деятельности может быть разделена на полосы потоков (swimlane) для объектов, которые показывают, какой объект и за какие виды деятельности ответственен. Виды деятельности (activity) представлены в виде прямоугольников с закругленными углами. Внутри описан вид деятельности: «вставить карту» (insert card), «ввести персональный идентификационный номер» (enter PIN) и т.п. Эти прямоугольники соединены единичными переходами. Разветвление (branch) и объединение (merge) переходов в пределах одного потока выполняются с помощью специальных вершин ветвления, они изображены в виде ромбов. Переходы могут также разветвляться на две и более **параллельно** выполняемые деятельности, то есть выполняемые в различных потоках. Параллельные ветвления (fork) и объединения (join) изображаются в виде темной горизонтальной линии.

Необходимо отметить, что при объединении параллельных переходов в вершинах типа JOIN переход из такой вершины активизируется только при всех активных входах (логическое “И”). В нашем примере: банковский автомат не может перейти к операции «показать остаток» (show balance) до тех пор, пока не выполнены две параллельные операции: 1) пока банк не произвел вычет получаемой суммы со счета клиента (debit account); 2) пока клиент не получил деньги из слота автомата (take money from slot).

2.9. Диаграммы компонентов и развертывания

Диаграмма компонентов показывает, какие компоненты есть в данной системе и какие между ними существуют зависимости. Компонентами системы мы называем отдельные программные блоки, из которых состоит вся система. Понимание зависимостей между компонентами дает возможность отслеживать на модели результаты изменений в отдельных компонентах. Помимо того, в этом представлении модели иногда указывается, с какими классами и элементами связан конкретный компонент.

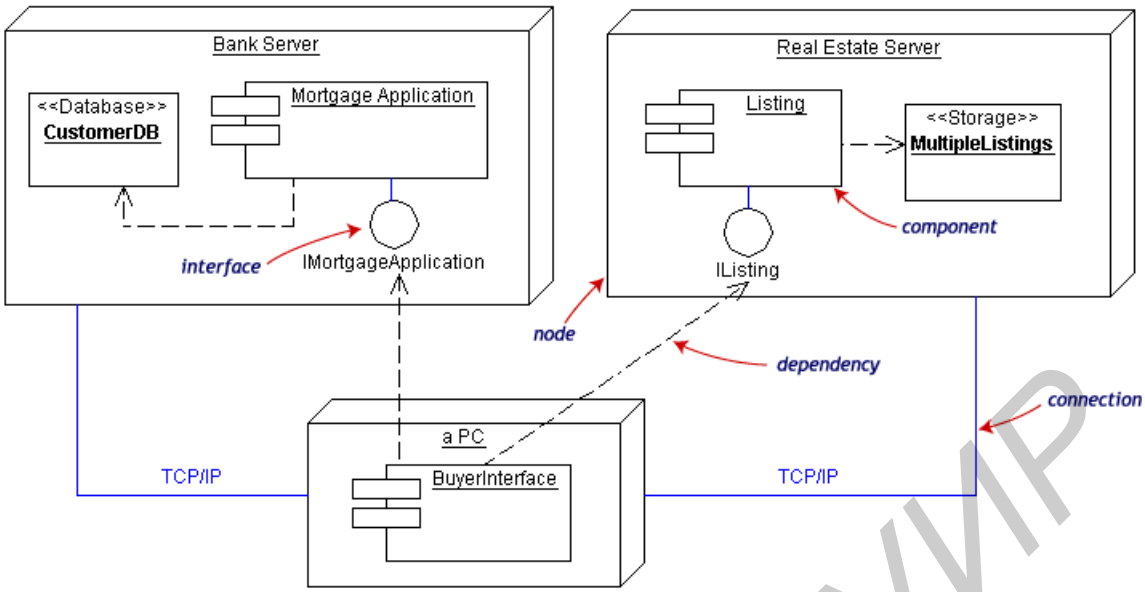

Рис.15. Диаграмма развертывания

Диаграмма развертывания (рис.15) отражает расположение работающих компонентов на узлах. Узел (node) — это ресурс, используемый во время выполнения программы (например, компьютер, аппаратное устройство или память). Это представление служит для изображения распределения ресурсов и их размещения.

3. CASE-система Rational Rose

Rational Rose — это CASE-система для визуального моделирования объектно-ориентированных программных продуктов. Визуальное моделирование - процесс графического описания разрабатываемого программного обеспечения. Моделирование системы позволяет обнаружить недостатки проекта ещё до написания кода, то есть на стадии, когда их исправление не требует значительных затрат.

Среда Rational Rose позволяет проектировать программную систему и строить модели с использованием всех видов диаграмм UML. С помощью моделей команда участников проекта сможет отслеживать реализацию исходных требований еще до этапа написания программного кода, а также из любого фрагмента кода выводить исходные требования, которые он реализует. Итак, Rose - это средство, которое может быть использовано всеми участниками проекта. Это, фактически, хранилище информации о контексте и проекте системы, из которого каждый участник проекта извлекает то, что ему нужно.

Помимо всего вышесказанного, Rational Rose позволяет генерировать "скелетный код" для нескольких языков, включая C++, Java, Visual Basic и PowerBuilder. Более того, можно выполнять обратное проектирование и на основе исходного кода получать модели для уже существующих систем. Если сделано изменение в модели, Rose позволяет модифицировать код для его реализаций. Если же был изменен код, можно автоматически обновить соответствующим образом и модель. Благодаря этому удается поддерживать соответствие между моделью и кодом, уменьшая риск "устаревания" первой.

3.1. Интерфейс Rational Rose

В этом подразделе описываются основные элементы интерфейса Rose. Это приложение управляется с помощью меню, однако существуют панели инструментов, облегчающие работу с чаще всего используемыми утилитами. Программа поддерживает работу с несколькими типами диаграмм UML: диаграммами вариантов использования, последовательности, кооперативными, классов, состояний и размещения. Для диаграмм каждого типа имеется соответствующая панель инструментов. Помимо панелей инструментов и меню Rose предлагает контекстные всплывающие меню, выводимые при щелчке правой кнопкой мыши. Например, щелчок правой кнопкой мыши на классе приведет к появлению меню с параметрами для изменения его атрибутов и операций, для просмотра или редактирования его спецификаций, для генерации, просмотра и редактирования соответствующего кода. Проще всего работать с Rose с помощью браузера. Это позволяет быстро и легко получать доступ к диаграммам и другим элементам модели.

3.1.1. Элементы экрана

Экран среды Rational Rose показан на рис.16. В его составе выделим шесть элементов: строку инструментов, панель «инструменты диаграммы», окно диаграммы, браузер, окно спецификации, окно документации. Кнопки строки инструментов позволяют выполнять стандартные и специальные действия. Содержание панели инструментов диаграммы меняется в зависимости от активной диаграммы. В окне диаграммы можно создавать, отображать и изменять диаграмму на языке UML.

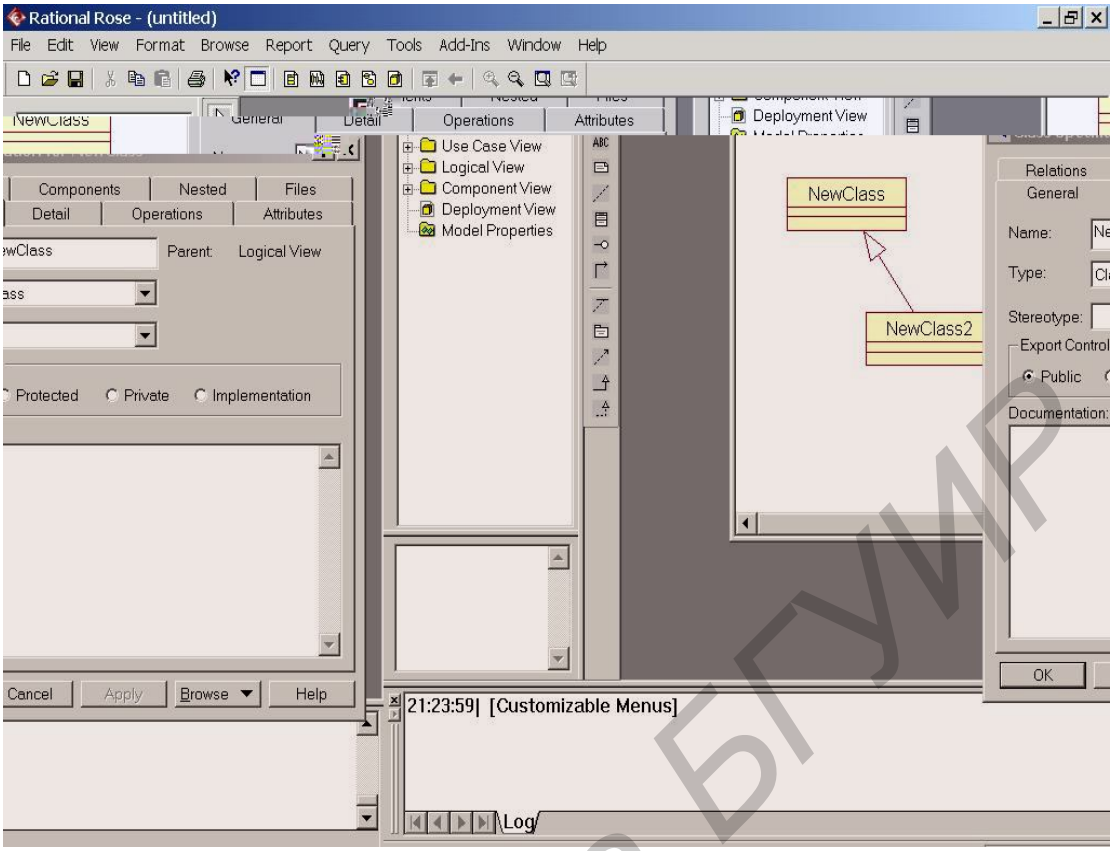

Рис. 16 . Интерфейс Rose

Браузер (browser) - используется для быстрой навигации по модели.

Окно документации (documentation window) - применяется для работы с документацией элементов модели.

Панели инструментов (toolbars) - обеспечивают быстрый доступ 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.

3.1.2. Браузер

Браузер — это иерархическая структура, позволяющая легко осуществлять навигацию по вашей модели. Все, что добавляется к модели: действующие лица, сценарии, классы, компоненты — выводится в окне браузера.

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 "+" — ветвь свернута.

Браузер поддерживает четыре представления (view): представление вариантов использования, компонентов, размещения и логическое представление.

С помощью браузера можно просматривать элементы модели в каждом из четырех представлений, перемещать и редактировать элементы, а также добавлять новые.

3.1.3. Окно документации

Это окно предназначено для документирования элементов модели Rose. Например, вы можете сделать короткое описание каждого действующего лица .

При документировании класса все, что вы напишете в окне документации, появится затем как комментарий в сгенерированном коде, что избавляет от необходимости впоследствии вносить комментарии вручную. Документация будет выводиться также в отчетах, создаваемых в среде Rose.

Если в браузере или на диаграмме выбирается другой элемент, окно документации автоматически обновляется, показывая то, что соответствует этому элементу.

3.1.4. Панели инструментов

Панели инструментов Rose обеспечивают быстрый доступ к наиболее распространенным командам. Предлагаются 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. Различные панели диаграмм подробно рассматриваются ниже.

Пользователь может изменить и настроить любую панель инструментов. Для этого следует выбрать пункт меню Tools / Options (Инструменты / Параметры), затем вкладку Toolbars (Панели инструментов).

Показать или скрыть стандартную панель инструментов можно следующим образом:

1. Выберите пункт Tools / Options (Инструменты / Параметры).

2. Выберите вкладку Toolbars (Панели инструментов).

3. Установите или сбросьте флажок Show Standard Toolbar (Показать стандартную панель инструментов).

Если нужно показать или скрыть панель инструментов диаграммы:

1. Выберите пункт Tools / Options (Инструменты / Параметры).

2. Выберите вкладку Toolbars (Панели инструментов).

3. Установите или сбросьте флажок Show Diagram Toolbar (Показать панель инструментов диаграммы).

Для увеличения размера кнопок панели инструментов:

1. Щелкните правой кнопкой мыши на требуемой панели.

2. Во всплывающем меню выберите пункт Use Large Buttons (Использовать большие кнопки).

Для настройки панели инструментов:

1. Щелкните правой кнопкой мыши на требуемой панели.

2. Выберите пункт Customize (Настроить).

Чтобы добавить или удалить кнопки, выберите соответствующую кнопку и затем щелкните мышью на кнопке Add (Добавить) или Remove (Удалить).

3.1.5. Окно диаграммы и журнал

В окне диаграммы выводится одна или несколько диаграмм UML вашей модели. При внесении изменений в элементы диаграммы 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.

По мере работы над моделью определенная информация направляется в окно журнала. Например, туда помещаются сообщения об ошибках, возникающих

при генерации кода. Невозможно закрыть журнал совсем, но его окно может быть минимизировано.

3.2. Работа в среде Rose

Все, что вы делаете в среде Rose, связано с моделями. В этом подразделе показано, как использовать модели.

3.2.1. Создание моделей

Первым шагом в работе с Rose является создание моделей. Их можно строить либо "с нуля", либо взяв за основу существующую каркасную модель. Готовую модель Rose со всеми диаграммами, объектами и другими элементами можно сохранить в одном файле, имеющем расширение . mdl (model).

Рис. 17. Мастер каркаса

Для создания модели:

1. Выберите в меню пункт File / New (Файл / Создать).

2. Если у вас установлен Мастер каркаса (Framework Wizard), то на экране появится список доступных каркасов (рис.17). Выберите каркас и щелкните на кнопке ОК. Если вы не планируете работать с каркасами, щелкните на кнопке Cancel (Отмена).

3.2.2. Сохранение моделей

Рекомендуется периодически сохранять файлы во время работы с ними. Вся модель сохраняется в одном файле. Кроме того, в отдельном файле можно сохранить журнал.

Для сохранения модели:

- выберите в меню пункт File / Save (Файл / Сохранить)

Или

- щелкните мышью на кнопке Save (Сохранить) стандартной панели инструментов.

3.2.3. Экспорт и импорт моделей

Одним из главных преимуществ объектно-ориентированной парадигмы является возможность повторного использования, применимая не только к коду, но и к самой модели. Для максимально полной ее реализации Rose поддерживает экспорт и импорт моделей и их элементов. Вы можете экспортировать модель или ее фрагменты и затем импортировать их в другие модели.

Для экспорта модели:

1. Выберите в меню пункт File / Export Model (Файл / Экспортировать модель).

2. Введите имя экспортируемого файла.

Для экспорта пакета классов:

1. На диаграмме классов выберите пакет, который нужно экспортировать.

2. Выберите в меню пункт File / Export <Package> (Файл / Экспортировать <пакет>).

3. Введите имя экспортируемого файла.

Для экспорта класса:

1. На диаграмме классов выберите класс, который нужно экспортировать.

2. Выберите в меню пункт File / Export <Class> (Файл / Экспортировать <класс>).

3. Введите имя экспортируемого файла.

Для импорта модели, пакета или класса:

1. Выберите в меню пункт File / Import Model (Файл / Импортировать модель).

2. Укажите файл, который требуется импортировать. Можно импортировать файлы моделей (.MDL), petal (.PTL), категорий (.CAT) и подсистем (.SUB).

3.2.4. Создание диаграммы вариантов использования

Для создания диаграммы Use Case (рис.18) нужно предварительно открыть соответствующее окно. Для этого необходимо:

1. В окне браузера развернуть папку Use Case View. Для этого следует щелкнуть по значку «+» слева от папки.

2. В развернутом дереве Use Case View выполнить двойной щелчок по значку Main. Открывается окно диаграммы вариантов использования и слева от окна открывается соответствующая панель инструментов.

Первый шаг в построении диаграммы состоит в определении действующих лиц (actors), фиксирующих роли внешних объектов, взаимодействующих с системой. Для добавления действующего лица в диаграмму необходимо:

1. На панели инструментов щелкнуть по значку Actor (изображение человечка).

2. Для добавления действующего лица в диаграмму щелкнуть в нужном месте диаграммы.

3. Пока добавленный значок остается выделенным, ввести имя нового действующего лица.

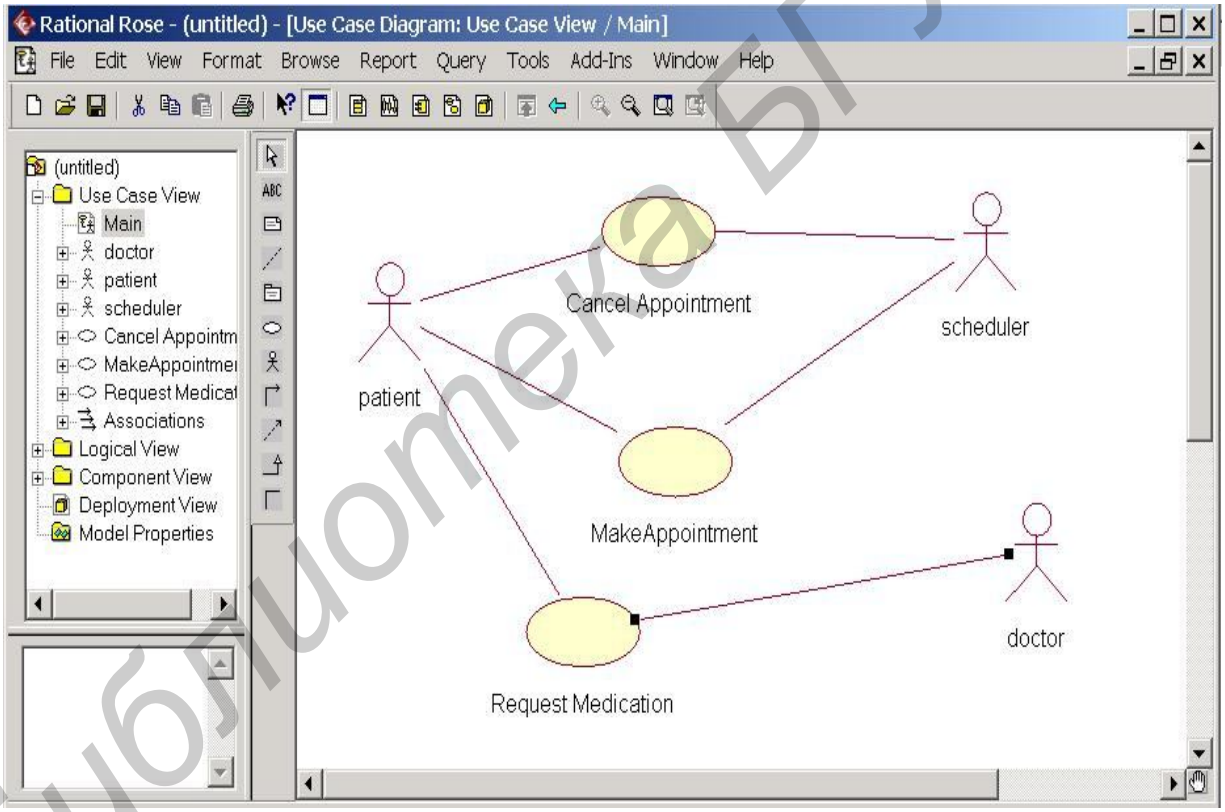

Рис. 18. Построение диаграммы вариантов использования в Rational Rose

На следующем шаге для каждого действующего лица нужно определить соответствующие варианты использования (use case), которые обозначают некоторую функциональность системы.

1. На панели инструментов щелкнуть по значку Use Case (изображение овала).

2. Для добавления варианта использования в диаграмму щелкнуть в нужном месте диаграммы.

3. Пока добавленный значок остается выделенным, ввести имя нового элемента Use Case.

4. Повторить первые три пункта для добавления остальных элементов Use Case.

Далее между действующими лицами и вариантами использования рисуются отношения.

1. На панели инструментов щелкнуть по значку ассоциации.

2. Для установления ассоциации между действующим лицом и вариантом использования в окне диаграммы щелкнуть по нужному значку действующего лица и перетащить линию на соответствующий элемент Use Case.

3.2.5. Создание диаграммы классов

Классы создаются в логическом представлении системы.

1. В окне браузера щелкнуть правой кнопкой по значку пакета Logical View.

2. В развернутом дереве Logical View выполнить двойной щелчок по значку Main. Открывается окно диаграммы классов и слева от окна открывается соответствующая панель инструментов.

3. Для добавления класса в диаграмму щелкнуть по значку Class на панели инструментов. Затем щелкнуть в необходимом месте окна диаграммы.

4. Пока добавленный класс остается выделенным, ввести имя.

5. Повторить эти операции для добавления остальных классов.

Добавление атрибутов и операций в класс можно выполнить двумя способами:

1. Щелкнуть правой клавишей по изображению класса на диаграмме. Появляется меню, в котором есть пункты «New Attribute» (добавить атрибут) и «New Operation» (добавить операцию).

2. Второй способ – это двойной щелчок левой кнопкой мыши по изображению класса. Появляется окно спецификации класса, в котором есть закладки «Attributes» и «Operations».

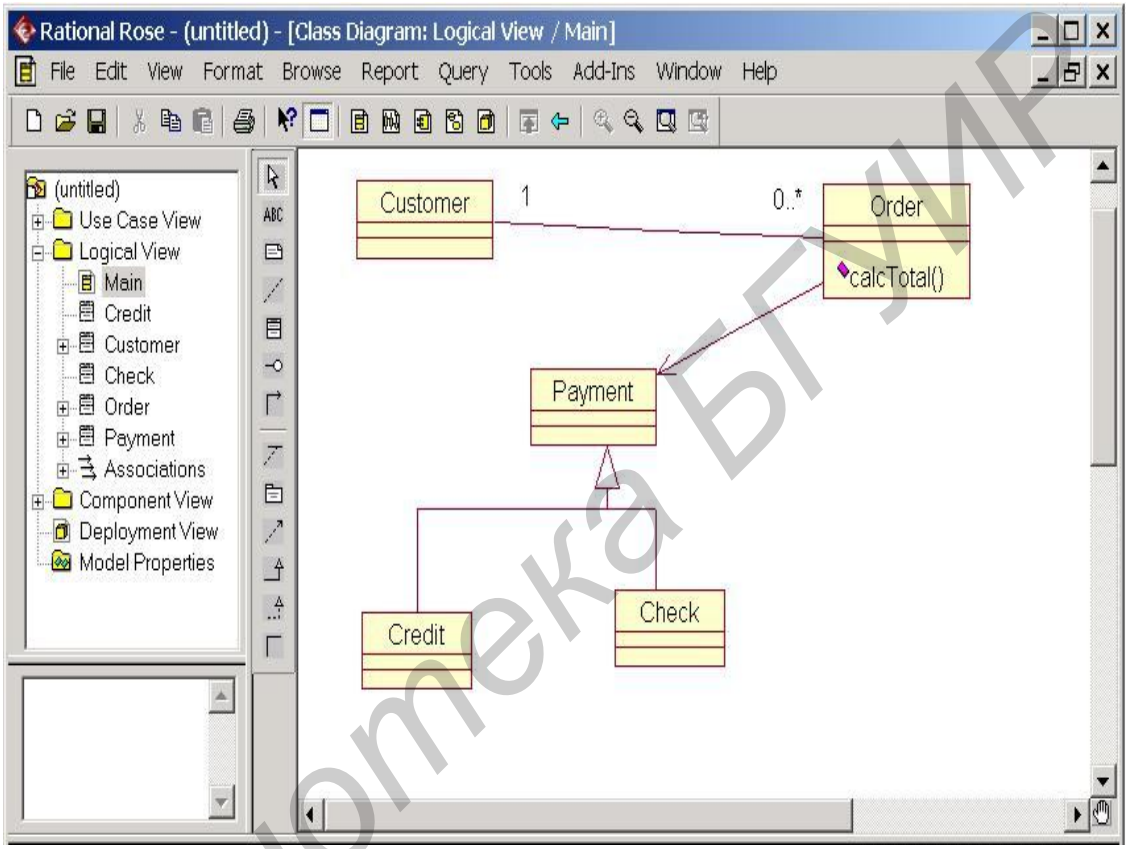

Рис. 19. Построение диаграммы классов в Rational Rose

После создания классов между ними устанавливаются связи. Отношения ассоциации, обобщения (наследования) и т.д. выбираются путем активизации соответствующих элементов панели инструментов.

Такие свойства отношений ассоциации и агрегации, как имя роли (role name), множественность (multiplicity), видимость (navigable), можно установить через соответствующее меню, которое вызывается путем шелчка правой кнопкой у конца линии отношения.

Литература

1. Рамбо Дж. и др. UML: Спец. справочник. СПб.: Питер, 2002.

2. Богс У., Богс М. UML и Rational Rose. М.: Лори, 2000.

3. Орлов С.А. Технологии разработки программного обеспечения. СПб.: Питер, 2002.

4. Ларман К. Применение UML и шаблонов проектирования. М: Вильямс, 2001.

5. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. 2-е изд. М.: Бином, 1998.

Св. план 2003, поз. 40

Учебное издание

Галковский Андрей Викторович

Применение языка UML при объектно-ориентированном проектировании

Методическое пособие

для студентов специальности
31 03 04 “Информатика”
дневной формы обучения

Редактор Т.А. Лейко
Корректор Е.Н. Батурчик

Подписано в печать 20.01.2004.	Формат 60x84 1/16.	Бумага офсетная.
Печать ризографическая.	Гарнитура «Таймс».	Усл.-печ.л. 2,33.
Уч.-изд.л. 1,7.	Тираж 100 экз.	Заказ 534.

Издатель и полиграфическое исполнение:
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»
Лицензия ЛП № 156 от 30.12.2002.
Лицензия ЛВ № 509 от 03.08.2001.
220013, Минск, П.Бровки, 6