МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО

ОБРАЗОВАНИЯ РФ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Т. В. ГЛОТОВА

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ МЕТОДОЛОГИЯ РАЗРАБОТКИ СЛОЖНЫХ СИСТЕМ

Учебное пособие

ПЕНЗА 2001

УДК 681.3

Г48

Приведено описание основ объектно-ориентированной методологии рассматриваемой в лекционном курсе "Разработка САПР". Учебное пособие содержит определения основных понятий - объектов, классов и отношений, методики объектно-ориентированного анализа, основные элементы объектного подхода. Рассматриваются составные части объектно-ориентированной методологии, основные этапы жизненного цикла при объектно-ориентированном подходе к разработке проекта, диаграммы унифицированного языка моделирования UML. Приведены сведения о наиболее распространенных CASE-средствах, поддерживающих объектно-ориентированную методологию разработки систем.

Учебное пособие разработано на кафедре "Системы автоматизации проектирования" и предназначены для студентов специальности 22.03 изучающих курс "Разработка САПР".

Ил. 9, библиогр. 5 назв.

Рецензенты: кафедра Вычислительных машин и систем Пензенского технологического института;

Заведующий кафедрой Прикладной математики и информатики Пензенского Государственного Университета д.т.н., профессор Линьков В.М.

1. Введение в объектно-ориентированную методологию разработки систем.

Объектно-ориентированная технология развивается в различных областях вычислительной техники как средство решения проблем связанных со сложностью создаваемых систем. Объектный подход применяется не только в программировании, но также в проектировании интерфейса пользователя, баз данных, баз знаний и даже компьютерной архитектуры. Смысл такого широкого подхода состоит в том, что он позволяет применить объектную ориентацию для решения всего круга проблем, связанных со сложными системами. В основе объектно-ориентированного проектирования лежит представление о том, что программную систему необходимо проектировать как совокупность взаимодействующих друг с другом объектов, рассматривая каждый объект как экземпляр определенного класса, причем классы образуют иерархию.

Повышение интереса разработчиков к этой методологии обусловлено тем, что методы структурного анализа и проектирования не обеспечивают дальнейшего снижения трудоемкости разработки. Объектно-ориентированный подход наиболее естественно соответствует реальному процессу разработки систем и не только программных, который является итеративным и может потребовать внести изменения в уже разработанные и отлаженные компоненты системы.

составными частями объектно-ориентированной методологии (ООМ) являются:

- объектно-ориентированный анализ;
- объектно-ориентированное проектирование;
- объектно-ориентированное программирование.

Объектно-ориентированное программирование. Обьектно-*ориентированное программирование — это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса, а классы образуют иерархию на принципах наследования.*

В данном определении можно выделить три части:

1) объектно-ориентированное программирование использует в качестве элементов конструкции объекты, а не алгоритмы;
2) каждый объект является реализацией определенного класса;
3) классы организованы иерархически.

Объектно-ориентированное проектирование. Методы программирования, прежде всего, подразумевают правильное и эффективное использование механизмов языков программирования. Методы проектирования напротив, основное внимание направляют на правильное и эффективное структурирование сложных систем.

Объектно-ориентированное проектирование — это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления как логической и физической, так статической и динамической моделей проектируемой системы.

В данном определении содержатся две важные части:

1) объектно-ориентированное проектирование ведет к объектно-ориентированной декомпозиции;
2) используется многообразие приемов представления моделей, отражающих логическую (структуры классов и объектов) и физическую (архитектура модулей и процессов) структуру системы.

Именно поддержка объектно-ориентированной декомпозиции отличает объектно-ориентированное проектирование от структурного проектирования.

Объектно-ориентированный анализ. На объектный подход оказали влияние предыдущие этапы развития программных средств. Традиционные приемы структурного анализа основаны на потоках данных в системе.

Объектно-ориентированный анализ (ООА) направлен на создание моделей, более близких к реальности, с использованием объектно-ориентированного подхода; это методология, при которой требования формируются на основе понятий классов и объектов, составляющих словарь предметной области.

На результатах ООА формируются модели, на которых основывается объектно-ориентированное проектирование; объектно-ориентированное проектирование в свою очередь создает основу для окончательной реализации системы с использованием методологии объектно-ориентированного программирования

Главными достоинствами ООМ по сравнению со структурными методами являются:

- возможность преодолеть ограничения, связанные со сложностью разрабатываемых систем;
- использование на стадии анализа моделей близких к реальности;

- применение как при анализе и проектировании информационных систем, так и систем реального времени и аппаратно-программных комплексов;
- обеспечение возможности повторного использования разработанного программного обеспечения, позволяющего существенно сократить сроки и снизить затраты на разработку каждой последующей системы;
- поддержка итеративного, а не лавинообразного, как в структурном подходе, процесса проектирования;
- естественная работа с разнородной информацией, используемой в мультимедиа системах;
- создание более открытых систем;
- полное использование описательных возможностей объектно-ориентированных языков программирования.

Принципы объектного подхода.

Объектная модель, которая является концептуальной базой объектно-ориентированной методологии, имеет четыре главных элемента:

- абстрагирование
- ограничение доступа или инкапсуляция
- модульность
- иерархия.

Без любого из этих элементов модель не будет объектно-ориентированной. Кроме главных имеется три дополнительных элемента:

- типизация
- параллелизм

- сохраняемость или устойчивость (persistence)

Эти элементы полезны в объектной модели, но не обязательны.

Абстрагирование - это выделение таких существенных характеристик объекта, которые отличают его от всех других видов объектов и таким образом чётко определяются особенности данного объекта с точки зрения дальнейшего его рассмотрения. Абстрагирование позволяет отделить самые существенные особенности поведения от несущественных. Абстракция определяет существенные характеристики некоторого объекта, которые отличают его от всех других видов объектов и четко очерчивает концептуальную границу объекта с точки зрения наблюдателя.

Выделяют целый спектр абстракций: абстракция сущности, поведения, абстракция виртуальной машины, произвольная абстракция.

Выбор достаточного множества абстракций, для заданной предметной области, является главной проблемой в объектном проектировании.

Инкапсуляция - это процесс разделения элементов объекта, определяющих его устройство и поведение; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации. В языке C++ управление доступом и видимостью достигается с большой гибкостью. Элементы объекта могут быть отнесены к общедоступной, обособленной и защищенной части. Инкапсуляция не спасает от глупости; она как заметил Страуструп защищает от ошибок, но не от жульничества.

Абстракция и ограничение доступа дополняют друг друга: абстрагирование направлено на наблюдаемое поведение объекта, а инкапсуляция занимается внутренним устройством. Практически это означает наличие двух частей в классе: интерфейса и реализации, Интерфейс отражает внешнее поведение объекта, описывая абстракцию поведения всех

объектов данного класса. Внутренняя реализация описывает представление этой абстракции и механизмы достижения желаемого поведения объекта. Разделение интерфейса и реализации позволяет защитить объекты от деталей реализации объектов более низкого уровня. Инкапсуляция позволяет вносить в программу изменения, сохраняя ее надежность и минимизируя затраты на этот процесс.

Модульность - это свойство системы, связанное с возможностью декомпозиции на ряд внутренне связанных, но слабо связанных между собой модулей. В языке C++ под модулями понимается раздельно компилируемые файлы.

Модульность - это разделение программы на раздельно компилируемые фрагменты, имеющие между собой средства сообщения. Традиционным в C++ является помещение интерфейсной части модулей в отдельные файлы с расширением .h.

Иерархия - ранжированная (упорядоченная) система абстракций. Основными видами иерархических структур, применительно к сложным системам, является структура классов (иерархия "is -a") и структура объектов (иерархия "part of"). Принцип наследования позволяет упростить выражения абстракции, делая проект менее громоздким и более выразительным.

Наследование - это такая иерархичность абстракций, в которой подклассы наследуют строение от одного или нескольких суперклассов. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Суперклассы отражают наиболее общие, а подклассы более специализированные абстракции. Поэтому о наследовании говорят, как об иерархии "обобщение специализации". Различают случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного суперкласса, во втором случае

суперклассов может быть несколько. Если в языке или системе поддерживается единичное наследование классов, набор классов образует древовидную иерархию. При поддержании множественного наследования классы связаны в ориентированный граф с корнем, называемый решеткой классов.

Иерархия по составу определяет отношения агрегирования. Более высокий уровень представляет те абстракции, которые используют в своём составе другие классы.

Принципы абстрагирования, ограничение доступа, иерархии конкурируют между собой. Принцип наследования требует открыть доступ к состоянию и к функциям объекта для производных объектов.

Дополнительные элементы:

Типизация - ограничение предъявляемых классу объектов, препятствующих взаимозамене различных классов и в большинстве случаев сильно сужающих возможность такой замены. Концепция типизации строится на понятии абстрактных типов данных. Тип - точное определение свойств строения или поведения, которое присуще некоторой совокупности объекта. Часто термины «тип» и «класс» считают эквивалентными. Более точно сказать, что класс реализует тип. Типизация- ограничение предъявляемых классу объектов, препятствующих взаимозамене различных классов и в большинстве случаев сильно сужающих возможность такой замены. Типизация позволяет выполнять описание абстракций т. о., что реализуется поддержка проектных решений на уровне языка программирования.

В тоже время объектно-ориентированные языки программирования могут быть: строго типизированными, нестрого типизированными и совсем не типизированными, что позволяет говорить о типизации, как о

второстепенном элементе. Сильно типизированные языки - это такие языки, в которых все выражения проводят проверку на соответствие типов. С++ поддерживает сильную типизацию. Различают статическую типизацию (раннее связывание) и динамическую типизацию (позднее связывание). Разделение имеет отношение ко времени, когда имена связывают с типами: статическая - во время компиляции; динамическая - во время исполнения программы.

Полиморфизм возникает на стыке принципов наследования и динамических связей. Это свойство является самым существенным в объектно-ориентированном программировании. Полиморфизм отличает объектно-ориентированное проектирование от более традиционных методов с использованием абстрактных типов данных.

Параллелизм. Для определенной категории задач автоматические системы реализуют обработку многих событий, происходящих одновременно. В то время как объектно-ориентированное программирование строится на абстракции, инкапсуляции и наследовании, параллелизм связан с абстрагированием процессов и синхронизацией. Объект является основой, которая объединяет обе концепции. Каждый объект (как абстракция реальности) может представлять собой отдельный поток управления (абстракцию процесса). Такой объект называется активным Параллелизм-свойство, отличающее активные объекты от пассивных. Для систем, построенных на основе объектно-ориентированного проектировании, реальность может быть представлена, как совокупность взаимодействующих объектов, часть из которых - активна.

Сохраняемость /устойчивость - это свойство объекта существовать во времени и/или пространстве, вне зависимости от процессов, породивших

данный объект. Выделяют следующие виды объектов, которые обладают различной степенью, сохраняем ости или устойчивости:

- Промежуточные результаты вычисления выражений.
- Локальные переменные вызова процедур.
- Собственные переменные (глобальные).
- Данные, сохраняющиеся между вызовами основной программы.
- Данные остающиеся без изменений в различных версиях программы.
- Данные, которые переживают создавшую их программу.

Традиционно языки программирования реализовывают первые три уровня, а последние три связываются с технологией БД. Введение сохраняемости приводит нас к объектно-ориентированным базам данных.

2. Классы и Объекты

2.1 Классы и объекты

Объект представляет собой особый опознаваемый предмет или сущность (реальную или абстрактную), имеющий четко определенное функциональное назначение в данной предметной области. Объект характеризуется состоянием, поведением, индивидуальностью. Структура и поведение одинаковых объектов описывается в общем для них классе. Термины «экземпляр класса» и «объект класса» взаимозаменяемы. Состояние объекта характеризуется перечнем всех его возможных свойств (обычно статических) и текущими значениями (обычно динамическими). Поведение объекта характеризуется изменением его состояний в процессе взаимодействия с другими объектами посредством передачи сообщения. Иначе поведение объекта полностью определяют его действие.

Операцией называется определенное воздействие одного объекта над другим, с целью вызова соответствующей реакции. Выделяют пять основных видов операций над объектами:

- Модификатор, изменяющий состояние объекта.
- Селектор, даёт доступ для определения состояния объекта без его изменения.
- Итератор, доступ к содержанию объекта по частям в определённой последовательности.
- Конструктор - создание и/или инициализация объекта.
- Деструктор - удаление объекта и/или освобождение занимаемой им памяти.

Индивидуальность - свойство, отличающее один объект от другого.

Сами по себе объекты не представляют интереса, цель системы реализуется только в процессе взаимодействие объектов между собой. Отношения между объектами: отношение двух любых объектов основывается на предположении, что каждый объект имеет информацию о другом объекте, операциях, которые над ним можно выполнить и об ожидаемом поведении.

Для объектно-ориентированного проектирования представляет интерес два типа отношений между объектами:

Отношение использование/ связь, которое, подразумевает возможность обмена сообщениями между объектами. В зависимости от того, как объект действует, он выполняет одну из трех ролей:

а) actor (актер, деятель), объект, воздействующий на другие объекты.

б) server (сервер), объект, подвергающий воздействию со стороны других объектов.

в) agent (агент), объект, который может быть как (1) так и (2).

Отношение включения/агрегации существует, когда один объект включает в себя другой объект. Т.е. один объект является элементом состояния другого объекта. По отношению к таким объектам применяются термины сложный, составной, агрегированный. Между отношениями включения и использования существует взаимосвязь. Включение одних объектов в другие предпочтительнее в том плане, что при этом уменьшается число объектов, с которыми приходится оперировать на данном уровне описания. С другой стороны, использование одних объектов другими имеет преимущество, так как не возникает сильной зависимости между объектами, как в случае включения.

Класс - множество объектов, связанных общностью структуры и поведением. Существует явное разделение внутреннего и внешнего описания класса. Интерфейсная часть описания класса соответствует его внешнему проявлению, подчёркивает его абстрактность, но скрывает структуру и особенности поведения.

Реализация составляет его внутреннее проявление и определяет особенности поведения. Интерфейсная часть описания класса может быть разделена на три составные части: общедоступную; защищённую и обособленную.

С точки зрения контрактного программирования класс - это генеральный контракт между абстракцией и всеми ее клиентами. Все обязательства класса выражены в его интерфейсе.

Состояние объекта задается в его классе через определение констант или переменных, помещаемых в его защищенной или закрытой части.

Классом называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой.

2.2 Отношение между классами.

Известно три основных типа отношений между классами:

- Обобщение/специализация(«is a»)
- Агрегирование объектов или отношение целое/часть(«part of»)
- Отношение ассоциативности отражает семантическую связь между классами, которые не связаны никакими другими типами отношений.

Языки программирования реализуют несколько общих способов для отражения трех типов отношений между классами. ОО языки реализуют в разных комбинациях следующие механизмы отношений классов:

Ассоциация (association), наиболее общий и неопределённый вид отношений. Обычно в процессе детализации проекта, детализация превращается в какую-то специализированную связь.

Наследование (inheritance) или обобщение (generalization) , наиболее эффективный тип отношений, который используется, как для отражения общности, так и для отражения ассоциативности.

Агрегация (aggregation), описание одного класса включает описание другого.

Зависимостью (dependence) называют отношение использования, согласно которому изменение спецификации одного элемента может повлиять на другой элемент его использующий. Чаще всего зависимости применяются при работе с классами, чтобы отразить тот факт, что один класс использует другой в качестве аргумента. В типичном случае такое отношение использования проявляет себя, если в реализации какой-либо операции происходит объявление используемого класса.

Инстанцирование (instantiation). Этот тип отношений, охватывает, так же обобщение и ассоциативность, но другим способом. В данном случае используется механизм конкретизации обобщения, за счёт использования классов-контейнеров, экземпляры которых состоят из наборов других объектов и могут быть, либо однородными, или неоднородными, состоящие из объектов разных классов, имеющие разный суперкласс. Возможно четыре основных способа построения таких классов:

а) Использование макроопределений.

б) Наследование и позднее связывание, создаёт однородные объекты.

в) Специальная процедура контроля типа, позволяющая в процессе образования объекта закрепить определённый класс элементов.

г) Параметризованный класс представляет собой шаблон для построения других классов, путём замены в базовом классе параметров на значения. Только после наполнения параметров такого класса значениями возможно создание объекта.

Метаклассы - это особый тип отношений, который реализуется не всеми объектно-ориентированными языками. Метакласс - это класс классов, позволяющих трактовать классы, как объекты.

Общие правила для определения типа отношений:

Если некоторая абстракция представляет нечто большее, чем сумма компонент, то лучше использовать отношения агрегации.

Если абстракция является подвидом другой абстракции, или соответствует простой сумме компонент, то следует использовать отношение наследования.

Взаимосвязь классов и объектов. В большинстве практических задач классы статичны, т. е. все их особенности и содержание определены в процессе компиляции программ. Объекты, в процессе выполнения программы, непрерывно создаются и разрушаются, т. е. объекты изменчивы и динамичны.

2.3 Вопросы качества при создании классов и объектов.

Для построения системы должен использоваться минимальный набор неизменяемых компонент, если компоненты должны быть структурированы и связаны единым способом построения. Для оценки качества классов и объектов, выделяемых в системе, предлагаются следующие пять критериев:

-взаимосвязанность (зацепление), степень глубины связи между отдельными модулями;

-связность, степень взаимодействия между элементами отдельного модуля (наиболее предпочтительна наибольшая связанность);

-достаточность, наличие в классе или модуле всего необходимого для реализации логичного и необходимого поведения;

- полнота, наличие в интерфейсной части класса всех необходимых характеристик абстракции. Существует формализованная описание абстракции и теория вывода полноты.

-простота/примитивность, все операции должны быть простыми.

3 Объектно-ориентированный анализ.

3.1 Задача классификации

Определение классов и объектов - одна из сложных задач объектно-ориентированного проектирования. Но, к сожалению, пока не разработаны строгие методы классификации и нет правила, позволяющего выделять классы и объекты. Но имеется богатый опыт классификации в других областях науки, на основе которого разработаны методики объектно-ориентированного анализа. Каждая методика предлагает свои правила идентификации классов и объектов.

Целью классификации является нахождение общих свойств в объектах. Классифицируя, мы объединяем в одну группу объекты, имеющие одинаковое строение или одинаковое поведение. Разумная классификация - часть любой точной науки. Классификация - последовательный и итерационный процесс. Трудность классификации обуславливается в основном широким выбором возможных равноправных решений. Исторически сложились три основных подхода к классификации:

- классическое распределение по категориям,
- концептуальная кластеризация,
- теория прототипов.

Классическое распределение по категориям (группирование по свойствам). В классическом подходе все вещи, обладающие данным свойством или совокупностью свойств, формируют некоторую категорию, то есть наличие свойства является основным критерием схожести объекта. При этом объекты можно разделить на непересекающиеся множества в зависимости от наличия или отсутствия некоторого признака. По предложению Мински М., наиболее подходящий набор свойств для такой

классификации характеризуется высокой независимостью этих свойств относительно друг друга. Этим объясняется такой популярный набор критериев как размер, цвет, форма, и материал. Свойства необязательно должны быть определены измеряемыми характеристиками, в качестве их можно использовать наблюдаемое поведение. Конкретные свойства, которые необходимо выделить при классификации определяются решаемой проблемой.

Концептуальная кластеризация (классификация по понятиям) возникла из попыток формального представления знаний. При таком подходе сначала формируется концептуальное описание классов (кластеров объектов) и затем объекты классифицируются согласно описанию, тем самым, образуя классы. Такое распределение объектов по классам имеет явно выраженные вероятностные свойства. Концептуальную кластеризацию можно связать с теорией нечетких (многозначных) множеств, в которой объект может принадлежать к нескольким категориям одновременно с разной степенью точности.

Теория прототипов относится к более современным методам классификации. Существуют некоторые абстракции, которые не имеют ни чётких свойств, ни чёткого определения (например, игры) и рассмотренные методы не работают. В теории прототипов класс определён одним объектом-прототипом и новый объект можно включить при условии, что он определён образом похож на прототип.

Эти три способа классификации составляют теоретические основы объектно-ориентированного анализа групп, которые мы можем применить для идентификации классов и объектов при проектировании сложной системы. На практике мы идентифицируем классы и объекты исходя, прежде всего из свойств рассматриваемой предметной области. Если с помощью

этого подхода не удается составить приемлемую структуру, приходится концептуально группировать объекты. Если и в этом случае не можем адекватно смоделировать задачу, то приходится прибегать к классификации с помощью ассоциативных методов, выделяя группы объектов по признаку сходства их с некоторым объектом прототипом.

3.2 Методики объектно-ориентированного анализа.

В процессе объектно-ориентированного анализа мы моделируем задачу, определяя классы и объекты, которые формируют словарь предметной области.

Классические подходы. Они основывается на классическом распределении по категориям.

Кандидаты для классов и объектов, предлагаемые С. Шлаером и С. Меллолором:

- материальные предметы
- роли (учитель, телезрители, и т. д.)
- события (прерывание, требование)
- взаимодействие (встреча, пересечение).

При моделировании баз данных Р. Росс прелагает свой аналогичный список:

-люди;

-места;

-предметы;

-организации;

-концепции;

-события;

Коад и Йордан предложили свой список кандидатов:

- структуры;
- другие системы;
- устройства;
- события;
- роли, в которых находятся пользователи;
- местоположение;
- организационные единицы.

Анализ поведения. В то время как классические подходы концентрируют внимание на осязаемых элементах предметной области, другое направление объектно-ориентированного анализа считает в качестве первоисточника объектов и классов динамическое поведение. Этот подход подобен концептуальной кластеризации: классы формируются, основываясь на группах объектов, имеющих сходное поведение. Предлагается понятие ответственности объекта, которое определяет его "знания и умения". Ответственность объекта - совокупность всех услуг, которые он может предоставлять по всем его контрактам. В иерархии классов каждый подкласс выполняет обязательства суперкласса и добавляет свои дополнительные услуги.

Анализ предметной области. Для поиска общих классов и объектов рекомендуется обратиться ко всем приложениям в рамках предметной области. Здесь выделяются те объекты, операции, связи, которые эксперты данной предметной области считают наиболее важными. В роли эксперта часто выступают просто пользователи системы.

Анализ вариантов (анализ сценариев). Классический подход, поведенческий подход и изучение предметной области по отдельности

сильно зависят от индивидуальных способностей и опыта аналитика. Анализ вариантов - это подход, который можно успешно сочетать с тремя первыми, делая их применение более упорядоченными. Этот вид анализа начинается вместе с анализом требований, когда пользователи, эксперты и разработчики перечисляют сценарии, наиболее существенные для работы с системой. Затем сценарии тщательно прорабатывается, раскладывается по кадрам. При этом устанавливается, какие объекты участвуют в сценарии, обязанности каждого объекта и как они взаимодействуют в терминах операций, т.е. четко распределяются области влияния абстракций. Далее набор сценариев расширяется, чтобы учесть исключительные ситуации и вторичное поведение. В результате появляются нов вые и уточняются существующие абстракции

CRC карточки. (Class-Responsibilities-Collaborators, Класс-Ответственность-Участники). Это простой и эффективный способ анализа сценариев. На карточке пишется карандашом сверху название класса, в левой половине - за что он отвечает, в правой - с кем сотрудничает. Проходя по сценарию, на каждый обнаруженный класс заводится по карточке. После анализа ответственности класса, возможно, часть ответственности с одного большого класса передается другому классу, или выделяются новые более детальные классы. Карточки можно раскладывать так, чтобы представить формы сотрудничества объектов. С точки зрения динамики сценария, их расположение показывает поток сообщений между объектами, с точки зрения статики они представляют иерархии классов.

Неформальное описание. В описание проблемы на обычном языке подчеркиваются существительные и глаголы. Существительные представляют собой кандидаты для классов; глаголы- кандидаты для

операций. Подход весьма приблизителен и не подходит для сложных проблем.

Структурный анализ. Возможно, использование структурного анализа для целей объектно-ориентированного проектирования, но этот подход не рекомендуется из-за опасности непроизвольно перейти к алгоритмической декомпозиции. Но если нет другой альтернативы и уже имеется модель системы, описанная диаграммами потоков данных. В результате анализа диаграмм потоков данных выделяют следующие кандидаты для объектов:

- внешние сущности;
- хранилища данных;
- хранилища управляющих сущностей.

Кандидаты для классов:

- потоки данных;
- потоки управления.

4. Система обозначений объектно-ориентированной методологии.

4.1 Язык UML.

Важный вопрос любой методологии - система обозначения, для визуального модулирования - графическая нотация для описания различных аспектов системы. Множество разработчиков предлагали свои варианты решения этого вопроса для объектно-ориентированной методологии. Наибольшую поддержку из них получили: нотация Буча, технология объектного моделирования OMT, разработанная Джеймсом Рамбо, объектно-ориентированное проектирование программного обеспечения OOSE Ивара Якобсона. В последствии эти три автора начали внедрять свои разработки идеи двух других, а затем начали работу по объединению этих методов в

компании Rational Software. Первая версия стандартной нотации UML появилась в январе 1997 года и большинство производителей ПО и производители CASE-средств поддерживают этот язык. В 1997 году группа OMG (Object Management Group) объявила UML промышленным стандартом. В настоящее время UML находится в процессе представления в качестве стандарта ISO.

UML это графический язык для специфицирования создания визуализации и документирования систем, в которых большая роль принадлежит программному обеспечению. С помощью UML можно разработать модель создаваемой системы, которая отображает не только ее концептуальные элементы, такие как функции системы и бизнесс-процессы ну и конкретные детали системы: классы языков программирования, схемы БД, повторно используемые компоненты ПО.

UML выделяет девять типов диаграмм. При рассмотрении статических аспектов системы используются:

- диаграммы классов;
- диаграммы объектов;
- диаграммы компонентов;
- диаграммы развертывания.

Для работы с динамическими частями системы применяются:

- диаграммы прецедентов;
- диаграммы последовательности;
- диаграммы кооперации;
- диаграммы состояний;
- диаграммы деятельности.

Диаграммы последовательностей и диаграммы кооперации также называются диаграммами взаимодействия.

В языке UML применяются четыре общих механизма: спецификация, дополнение (adornments), принятые деления (common divisions) и механизмы расширения (extensibility mechanisms).

Самой важной разновидность дополнения являются примечания, которые представляют собой графические символы для изображения ограничения или комментариев. Они используются для включения в модель дополнительной информации. В UML заложена открытость, т.е. возможность расширять язык контролируемым способом. К механизмам расширения UML относятся:

стереотипы (stereotype) предназначенные для расширения словаря UML (например, стереотип класса);

помеченное значение (tagged value) позволяет включать новую информацию в спецификацию элемента;

ограничение (constraint) позволяет добавлять или изменять существующие правила.

Спецификация - это неграфическая форма, используемая для полного описания элемента системы обозначений: класса, объекта, операции, диаграммы в целом. Большинство параметров класса определяется в спецификации. Спецификация класса содержит: имя, текстовое описание, стереотип, атрибуты, операции, ограничения, мощность или множественность класса (cardinality), параметры для параметризированного класса, сохраняемость/устойчивость (мгновенная, долговременная), параллельность (последовательная, охраняемая, синхронная, активная), объём памяти. При последовательной

параллельности (sequential) гарантируется нормальное поведение класса только при наличии одного потока управления. Охраняемый (guarded) класс - при наличии нескольких потоков, обращения к его операциям должны быть упорядочены, т.е. только одна операция в один момент времени. Синхронный класс может сам обрабатывать взаимные исключения. Активный класс будет иметь свой поток управления.

Спецификация операций содержит: имя, текстовое описание, класс возвращаемого значения, аргументы, стереотип, квалификация (является ли функция виртуальной, статической), видимость (открытая, защищенная, реализация), предусловие, действие, постусловие, ограничения, параллельность (последовательная, охраняемая, синхронная), время выполнения, объём памяти требуемой операции во время выполнения.

При моделировании объектно-ориентированных систем используются два подхода к делению реальности. Прежде всего, существует разделение на классы и объекты. Практически все строительные блоки UML характеризуются дихотомией класс/объект. В графическом представлении для объекта принято использовать тот же символ, что и для его класса, а название объекта подчеркивать. Второй вариант деления - это разделение на интерфейс и его реализацию. Интерфейс декларирует контракт, а реализация представляет конкретное воплощение этого контракта и обязуется точно следовать объявленной семантике интерфейса. Интерфейс применяется для моделирования стыковочных узлов.

Интерфейсом называется набор операций, используемый для определения услуг, предоставляемых классом или компонентом, выполняемых прецедентом или подсистемой. Интерфейс изображается в виде круга, присоединенному к реализующему его классу или компоненту. Интерфейс может быть изображен также как стереотипный класс с

множеством операций. Для интерфейса определена операция реализации. Он может быть реализован несколькими классами и наоборот один класс может реализовывать несколько интерфейсов. Отметим, что у интерфейса нет атрибутов и нет непосредственных экземпляров.

4.2 Основные диаграммы языка UML

Диаграмма прецедентов (use case) или вариантов использования показывает совокупность прецедентов и действующих лиц (actor), а также отношения между ними. Прецеденты - это не зависящее от реализации высокоуровневое представление того, что пользователь ожидает от системы, т.е. описание функциональности системы. Действующее лицо - это все, что взаимодействует с создаваемой системой. Варианты использования и действующие лица определяют сферу применения создаваемой системы. При этом прецеденты описывают все то, что происходит внутри системы, а действующие лица - то, что происходит снаружи. На языке UML действующие лица представляются в виде значков фигур, а варианты использования - в виде овалов (Рис. 1).

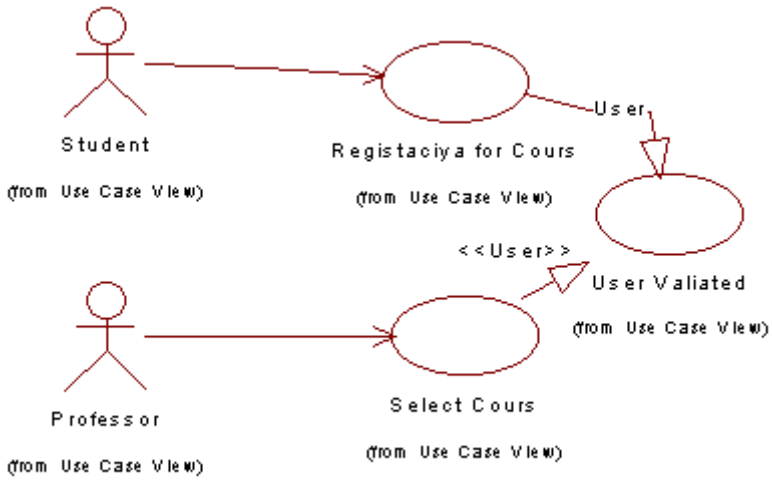


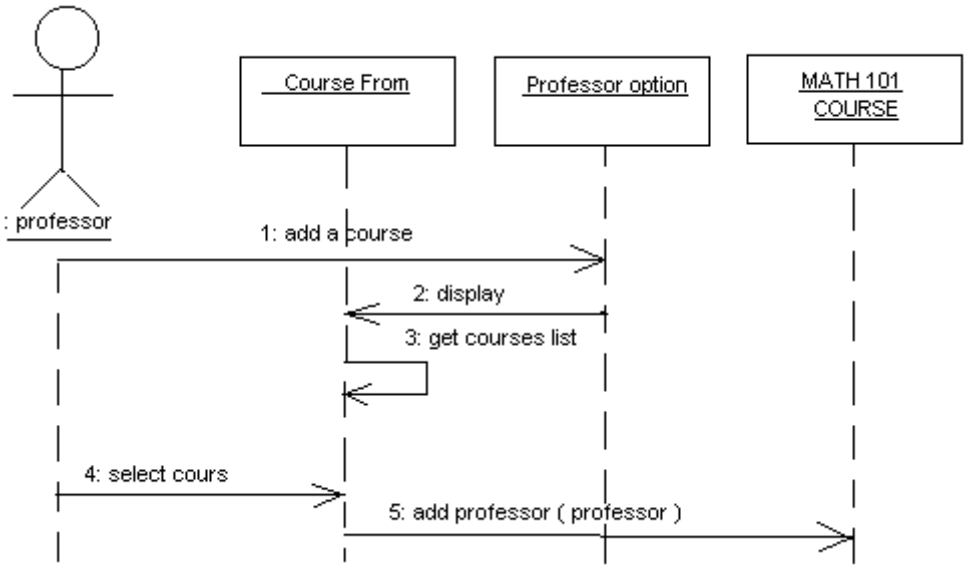
Рис. 1 Диаграмма прецедентов

В UML для вариантов использования и действующих лиц поддерживается несколько типов связей:

- связи коммуникации описывают связи между действующими лицами и вариантами использования,
- связи расширения (extends) и использования (uses) отражают связи между вариантами использования,
- обобщения действующего лица описывают связи между действующими лицами.

Диаграмма взаимодействия (Interaction diagram) описывает взаимодействия, состоящие из множества объектов и отношений между ними, включая сообщения, которыми они обмениваются.

Диаграммой последовательностей (Sequence diagram) называется диаграмма взаимодействия, акцентирующая внимание на временной упорядоченности сообщений. Графически такая диаграмма представляет собой таблицу, объекты в которой располагаются вдоль оси X, а сообщения в порядке возрастания времени - вдоль оси Y (рис.2).



На диаграмме последовательностей показаны линии жизни объектов и фокус управления (активизации объекта). Обычно для каждого варианта использования создаются несколько диаграмм последовательностей. Одна для варианта сценария без ошибок, другие отражают ход событий в альтернативных потоках. Отправление сообщений, которые могут обозначать событие или вызовы операций показываются горизонтальными стрелками с обозначением синхронизации. Линия, обозначающая посылку сообщения, проводится от вертикали клиента к вертикали сервера. Первое сообщение - на самом высоком уровне, второе ниже, и т.д. Сообщение на диаграмме показывает, что один объект вызывает функцию другого. В дальнейшем, когда будут определены операции классов, каждое сообщение станет операцией. Сообщения могут быть рефлексивными, что соответствует обращению объекта к своей собственной операции. На диаграмме взаимодействия можно добавить пояснения.

Диаграммой кооперации (Collaboration diagram) называется диаграмма взаимодействий, основное внимание, в которой уделяется структурной организации объектов, принимающих и отправляющих сообщения. Графически такая диаграмма представляет собой ориентированный граф с объектами в качестве вершин и сообщениями в качестве дуг. Кооперативная диаграмма показывает, какие объекты взаимодействуют друг с другом. Она полезна, когда нужно оценить последствия сделанных изменений.

С помощью диаграмм взаимодействия проектировщики и разработчики системы могут определить классы, которые нужно создать, связи между ними, а также операции и ответственности каждого класса.

Диаграмма объектов показывает, какие существуют объекты и связи между ними в логической структуре системы. Используется для предоставления сценария, т. е. диаграмма объектов - мгновенный снимок

потока событий, в некоторой конфигурации объекта. Диаграмма объектов используется при анализе, для показа семантики основных и второстепенных сценариев поведения системы, и при проектировании используется для показа семантики механизмов.

Диаграммы классов показывают, какие существуют классы и связи между ними в логической структуре системы. Класс обозначается прямоугольником, в котором указываются имя класса, его атрибуты и операции (рис.3). На диаграмме можно также отобразить стереотип класса, видимость всех атрибутов и операций, тип данных атрибутов и сигнатуру всех операций.

Рис.3 Изображение класса

Отношения между классами:

Ассоциация изображается в виде обыкновенной линии и может быть двунаправленной и однонаправленной. Ассоциация дает классу возможность узнавать об общих атрибутах и операциях другого класса. Рядом со знаком ассоциации может указываться мощность, которая ставится у конца линии ассоциации и означает число связей между каждым экземпляром класса в начале линии, с экземпляром класса в её конце. Ассоциация может быть рефлексивной. Рефлексивная ассоциация предполагает, что один экземпляр класса взаимодействует с другими экземплярами этого же класса. После определения ассоциации при генерации кода в классы помещаются дополнительные атрибуты, которые имеют тип ассоциируемого класса.

Наследование на языке UML называют обобщением и изображают в виде стрелки от подкласса к суперклассу (рис.4). Циклы запрещаются.

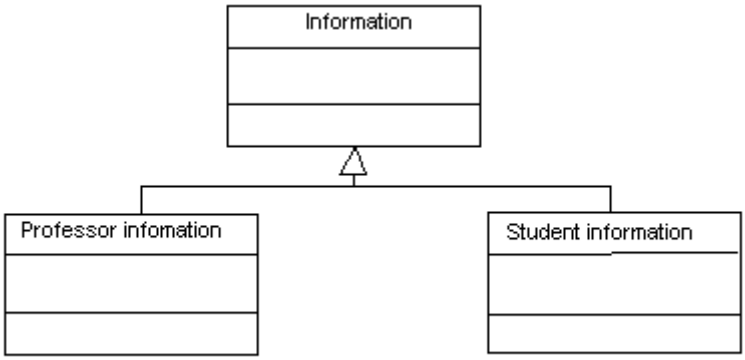


Рис.4 Диаграмма классов. Отношение наследования.

Агрегация представляет собой более тесную форму ассоциации. Это связь между целым и его частями. Агрегацию изображают в виде линии с ромбиком у класса, являющегося целым. При генерации кода для агрегации автоматически создаются поддерживающие ее дополнительные атрибуты.

Зависимости изображается в виде стрелки, проведенной пунктирной линией. Зависимости всегда однонаправленные, они показывают, что один класс зависит от определений, сделанных в другом. При генерации кода, специальные атрибуты для классов, связанных зависимостью не создаются.

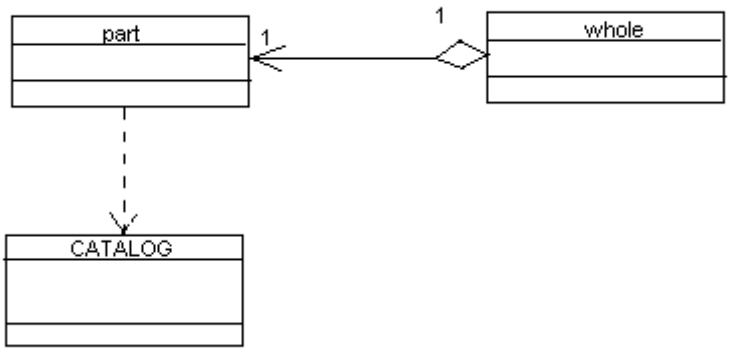


Рис.5 Диаграмма классов. Отношения агрегации и зависимости.

Когда система разрастается, то можно выделить группы классов сильно связанных внутри и слабее с другими. Для группирования классов, обладающих некоторой общностью в UML применяют пакеты. Наиболее часто классы группируются по их функциональности или по стереотипу.

Параметризованные классы изображаются значком обычного класса с пунктирным прямоугольником в правом верхнем углу, в котором указываются параметры.

Инстанцированный класс изображается обычным значком класса с прямоугольником со сплошной границей и перечислением в нём фактических параметров. Связь между ними отображается пунктирной линией со стрелкой, указывающей на параметризированный класс. Для получения инстанцированного класса необходим другой конкретный класс, как фактический параметр. Параметризированный класс не может порождать экземпляры и не может использоваться сам в качестве параметра.

Правила моделирования отношений в UML:

- используйте зависимость, только если моделируемое отношение не является структурным,
- используйте обобщение, если имеет место отношение типа "является" ("is a"),
- множественное наследование часто можно заменить агрегированием,
- иерархия наследования не должна быть слишком глубокой (не более пяти уровней), не слишком широкой (лучше прибегнуть к промежуточным абстрактным классам),
- применяйте ассоциации, прежде всего там, где между объектами существуют структурные отношения.

Диаграмма состояний показывает пространство состояний отдельного класса; события, которые влекут переход из одного состояния в другое; действия, которые происходят при изменении состояния. Отдельная диаграмма представляет собой динамическую модель данных отдельного класса, нескольких, наиболее существенных классов, или поведение системы в целом. Эта диаграмма используется в ходе анализа, чтобы показать динамику поведение системы, и в ходе проектирования - для выражения поведения отдельных классов или их взаимодействия. Диаграмма состояний показывает автомат, представляющий поток управления от состояния к состоянию. Начальное состояние обязательно присутствует на диаграмме и только одно, присутствие конечного состояния не обязательно и может быть несколько конечных состояний (рис.6).

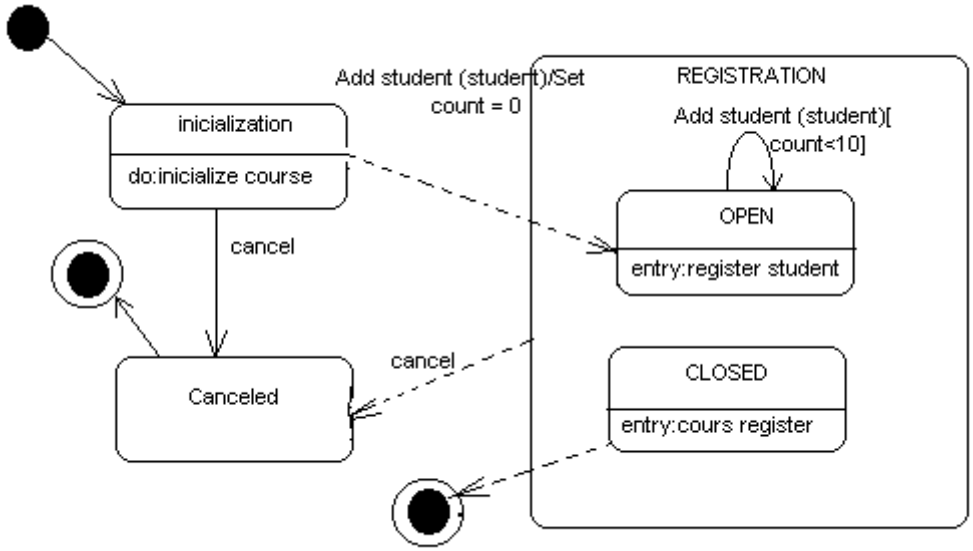


Рис.6 Диаграмма состояний

С помощью таких диаграмм удобно моделировать динамику поведения класса. Как правило, диаграммы состояний не требуется создавать для каждого класса, во многих проектах они вообще не используются. Если объект класса может существовать в нескольких состояниях и в каждом из

них ведет себя по-разному, то для такого класса, скорее всего, потребуется диаграмма состояний.

Диаграмма деятельности - диаграмма, на которой представлены переходы потока управления от одной деятельности к другой. Диаграмма относится к динамическому аспекту поведения системы. Это разновидность диаграмм состояний, где все или большая часть переходов срабатывают при завершении деятельности в исходное состояние. Примеры состояния деятельности: выполнение операции над объектом, создание, уничтожение. Диаграмма деятельности - это своеобразная блок-схема, которая описывает последовательность выполнения операций во времени.

Диаграмма компонентов - диаграмма, на которой изображено множество компонентов и зависимости между ними; относится к статическому виду системы. Компонент - это физически заменяемая часть системы, совместимая с одним набором интерфейсов и обеспечивающая реализацию какого-либо другого интерфейса. Компонент изображается в виде прямоугольника с вкладками (рис. 7)

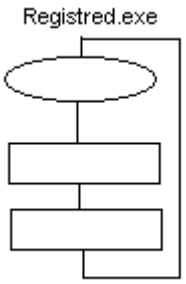


Рис. 7 Изображение компонента

К имени компонента обычно добавляется расширение имен файлов. Отношения между компонентами - это компиляционные зависимости, которые изображаются стрелкой, выходящей из зависимого модуля (рис. 8). Для определения различных типов компонентов используется механизм

стереотипов. Диаграммы компонентов могут также содержать пакеты или подсистемы для группирования элементов модели в более крупные блоки.

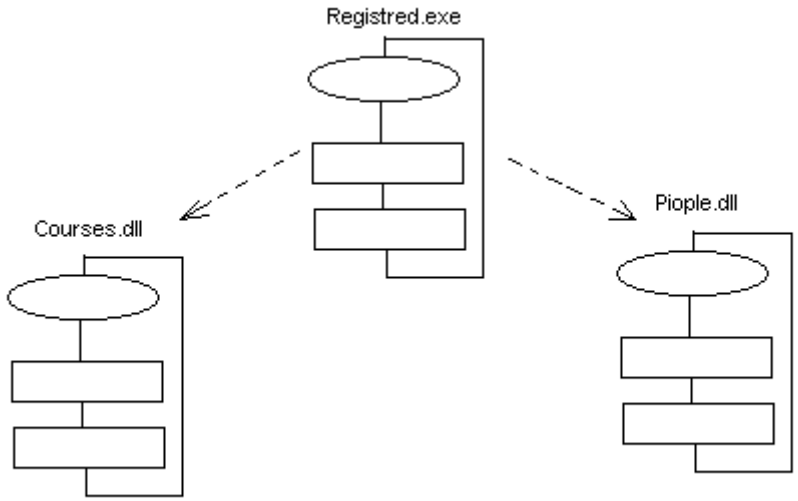


Рис.8 Диаграмма компонентов

Диаграмма компонентов дает представление о том, в каком порядке нужно компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. С помощью этой диаграммы можно оценить последствия любых вносимых изменений, можно определить какие части системы можно использовать повторно (чем от меньшего числа компонентов зависит компонент, тем легче его будет использовать повторно).

Диаграмма развертывания или размещения - это диаграмма, на которой представлена конфигурация обрабатывающих узлов и размещенные на них компоненты; относится к статическому виду системы. Пример диаграммы развертывания приведен на рис. 9.

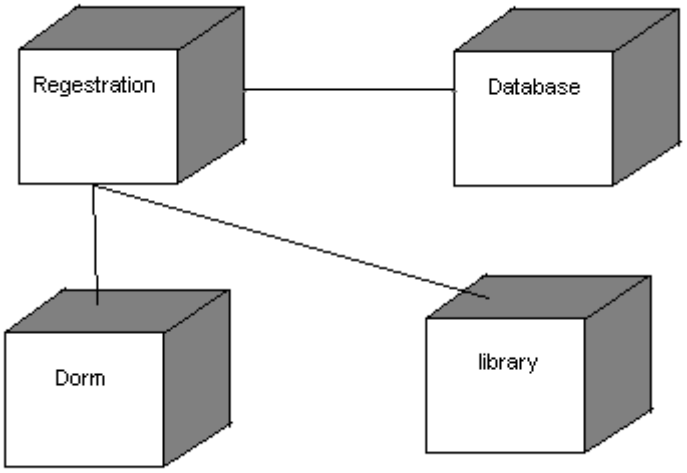


Рис.9 Диаграмма развертывания

5. Объектно-ориентированные CASE-системы

5.1 Общие сведения и классификация CASE-средств.

В последние годы на мировом рынке программных продуктов появилось много программных средств, называемых CASE-системами, CASE-инструментами или CASE-средствами (CASE-tools). Это программные системы, позволяющие автоматизировать использование новых методов проектирования ПО для разработки сложных систем на всех этапах разработки (особенно на ранних стадиях разработки). Термин CASE трактуется весьма широко. Первоначально он расшифровывался как Computer Aided Software Ingeneering (компьютерная поддержка проектирования ПО), и связано это было с автоматизацией разработки программных систем. В дальнейшем понятие CASE приобрело новый смысл, все чаще его стали расшифровывать как Computer Aided System Ingeneering (компьютерная поддержка проектирования систем), а

CASE-средства все больше стали ориентироваться на создание спецификаций, проектирование и моделирование сложных систем широкого назначения.

Применение CASE-продуктов позволяет отслеживать процесс принятия решений при разработке больших проектов, систематизировать информацию о проекте и его компонентах, упрощая тем самым верификацию проекта и сопутствующей ему документации. Важным аспектом CASE-подхода является поддержка коллективной работы, при которой каждый из разработчиков создает свою подсистему, причем в любой момент времени эти подсистемы могут быть объединены в общий проект.

Современные CASE-средства охватывают обширную область поддержки многочисленных технологий проектирования программных систем: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл ПО. Наиболее трудоемкими этапами разработки являются этапы анализа и проектирования, в процессе которых CASE-средства обеспечивают качество принимаемых технических решений и подготовку проектной документации. При этом большую роль играют методы визуального представления информации. Это предполагает построение структурных или иных диаграмм в реальном масштабе времени, использование многообразной цветовой палитры, сквозную проверку синтаксических правил. Графические средства моделирования предметной области позволяют разработчикам в наглядном виде изучать существующую ИС, перестраивать ее в соответствии с поставленными целями и имеющимися ограничениями.

Применение CASE-продуктов требует от потенциальных пользователей специальной подготовки и обучения. Опыт показывает, что внедрение этих продуктов осуществляется медленно. Однако по мере приобретения пользователями практических навыков и повышения общей культуры проектирования эффективность этих средств резко возрастает.

Если термин CASE понимать буквально, то CASE-системой можно считать всякую программную систему, помогающую в разработке программ, включая любой транслятор или систему программирования. Но CASE-системы, первоначально появившиеся на рынке программных продуктов под этим названием - это системы, поддерживающие этап анализа проектируемой системы и фиксацию результатов этой работы в виде соответствующих спецификаций.

Обычно к CASE-средствам относят любое программное средство, автоматизирующее ту или иную совокупность процессов жизненного цикла ПО и обладающее следующими основными характерными особенностями:

- мощные графические средства для описания и документирования системы, обеспечивающие удобный интерфейс с разработчиком и развивающие его творческие возможности;
- интеграция отдельных компонент CASE-средств, обеспечивающая управляемость процессом разработки системы;
- использование специальным образом организованного хранилища проектных метаданных (репозитория).

По мере того как слово CASE входило в моду, им стали называть самые разные инструментальные программные средства, относящиеся к автоматизации проектирования и разработкам программ. Поэтому необходимо ввести некоторую классификацию CASE-систем.

Прежде всего, CASE-системы классифицируются по уровням или этапам разработки программ, которые они охватывают, т.е. содержат средства, помогающие в работах, относящихся к этим уровням или этапам. Различают верхние (upper) и нижние (lower) системы.

Верхние CASE-системы поддерживают работы по уточнению постановки задачи и анализу проектируемых систем, в ходе которых составляются, корректируются и анализируются спецификации систем. Так как верхние CASE-систем поддерживают те же виды работ, что и упомянутые выше первоначальные CASE-системы, их называют еще нормальными.

Нижние CASE-системы поддерживают работы по проектированию программ, следующие за системным анализом, а также в той или иной степени собственно построение программ (кодирование, генерация кода). Границу между верхними или нижними CASE-системами проводят и несколько иначе, а также выделяют средние (middle) CASE-системы, поддерживающие уровень, промежуточный между верхним и нижним и частично пересекающийся с ними. Для верхних CASE-систем характерно использование графических средств, позволяющих строить, преобразовывать и анализировать разные виды диаграмм, сетей, деревьев. Появление первых CASE-систем связано с развитием графических средств на персональных компьютерах и рабочих станциях. В этом смысле они как системы автоматизации проектирования программ подобны системам автоматизации проектирования промышленных изделий (САПР), в которых главенствующую роль играют графические средства. Специфика нормальных CASE-систем, отличающая их от традиционных сред программирования и средств разработки программ, состоит в поддержке верхних уровней поддержки программ.

Все современные CASE-средства могут быть классифицированы в основном по типам и категориям. Классификация по типам отражает функциональную ориентацию CASE-средств на те или иные процессы ЖЦ.

Классификация по типам в основном совпадает с компонентным составом CASE-средств и включает следующие основные типы:

- средства анализа (Upper CASE), предназначенные для построения и анализа моделей предметной области (Design/IDEF (Meta Software), BPwin (Logic Works));
- средства анализа и проектирования (Middle CASE), поддерживающие наиболее распространенные методологии проектирования и использующиеся для создания проектных спецификаций (Vantage Team Builder (Cayenne), Designer/2000 (ORACLE), Silverrun (CSA), PRO-IV (McDonnell Douglas), CASE.Аналитик (МакроПроджект)). Выходом таких средств являются спецификации компонентов и интерфейсов системы, архитектуры системы, алгоритмов и структур данных;
- средства проектирования баз данных, обеспечивающие моделирование данных и генерацию схем баз данных (как правило, на языке SQL) для наиболее распространенных СУБД. К ним относятся ERwin (Logic Works), S-Designor (SDP) и DataBase Designer (ORACLE). Средства проектирования баз данных имеются также в составе CASE-средств Vantage Team Builder, Designer/2000, Silverrun и PRO-IV;
- средства разработки приложений. К ним относятся средства 4GL (Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Gupta), Delphi (Borland) и др.) и генераторы кодов, входящие в состав Vantage Team Builder, PRO-IV и частично - в Silverrun;

- средства реинжиниринга, обеспечивающие анализ программных кодов и схем баз данных и формирование на их основе различных моделей и проектных спецификаций. Средства анализа схем БД и формирования ERD входят в состав Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin и S-Designor. В области анализа программных кодов наибольшее распространение получают объектно-ориентированные CASE-средства, обеспечивающие реинжиниринг программ на языке C++ (Rational Rose (Rational Software), Object Team (Cayenne)).

- Вспомогательные типы включают:

- средства планирования и управления проектом (SE Companion, Microsoft Project и др.);

- средства конфигурационного управления (PVCS (Intersolv));

- средства тестирования (Quality Works (Segue Software));

- средства документирования (SoDA (Rational Software)).

Классификация по категориям определяет степень интегрированности по выполняемым функциям и включает отдельные локальные средства, решающие небольшие автономные задачи (tools), набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла ИС (toolkit) и полностью интегрированные средства, поддерживающие весь ЖЦ системы и связанные общим репозитарием (workbench).

Существует классификация CASE-средств по поколениям. Первое поколение характеризуется наличием разобщенных средств, повышающих производительность труда и улучшающих качество проектирования на отдельных этапах или операциях разработки. В настоящее время реализация

CASE-средств направлена на создание интегрированной среды комплексной автоматизации процессов проектирования, разработки и сопровождения, реализующих некоторую методологию проектирования ПС. Это направление названо второй генерацией CASE. В средствах этой генерации охватываются не только традиционные процессы проектирования и разработки, но и работы по анализу готового ПС (re-engineering) с целью устранения ошибок и, главное, оптимизации характеристик, а также и по укрупненному описанию готового ПС с целью проектирования нового по прототипу уже созданного (reverse engineering). Эти процессы, в первую очередь, традиционны для сопровождения и модификации ПС. Средства второго поколения, как правило, ориентированы на решение задачи комплексной автоматизации процесса разработки и сопровождения систем.

Помимо этого, CASE-средства можно классифицировать по следующим признакам:

- применяемым методологиям и моделям систем и БД;
- степени интегрированности с СУБД;
- доступным платформам.

Интегрированное CASE-средство (или комплекс средств, поддерживающих полный ЖЦ ПО) содержит следующие компоненты:

- репозиторий, являющийся основой CASE-средства. Он должен обеспечивать хранение версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль метаданных на полноту и непротиворечивость;

- графические средства анализа и проектирования, обеспечивающие создание и редактирование иерархически связанных диаграмм (DFD, ERD и др.), образующих модели ИС;
- средства разработки приложений, включая языки 4GL и генераторы кодов;
- средства конфигурационного управления;
- средства документирования;
- средства тестирования;
- средства управления проектом.

Нормальные CASE-системы могут содержать не все эти средства (например, может не быть средств прототипирования), и возможности средств того или иного вида могут быть представлены в них по-разному (например, графический редактор только для одного типа диаграмм или же для нескольких типов).

Обычная схема реализации upper CASE состоит в том, что сердцем такого средства является объектно-ориентированное хранилище (repository), доступ к которому имеют все подсистемы. Хранилище содержит сведения о каждом элементе проекта отдельно вне зависимости от способа их получения: из графического редактора или таблиц. Общую схему современного upper CASE можно охарактеризовать наличием подсистем для ввода и проверки моделей, управления конфигурацией проекта, интерфейсов с СУБД и языком четвертого поколения и генерации кода (включая средства реинжиниринга), а также средствами документирования проекта и средствами коммуникаци с другими upper CASE.

Направления развития CASE-средств определяются потребностями практики и ориентированы на:

- расширение применяемых моделей описания автоматизируемых систем;
- охват автоматизацией новых архитектур ИС, включая схему клиент/сервер;
- более глубокий уровень контроля целостности проекта;
- интеграцию со многими 4GL, СУБД и CASE;
- использованием новых платформ, прежде всего рабочих станций.

CASE системы позволяет существенно ускорить не только разработку системы, но также и ее сопровождение, так как внесение изменений существенно упрощается, поскольку информация о разработке хранится в базе данных CASE системе и может быть легко модифицирована. По некоторым данным использование CASE в процессе разработки позволяет сократить ее сроки в 3 раза, а сроки внесения изменений в 200 раз.

Проекты средней, высокой сложности и уникальные рекомендуется создавать с помощью CASE-средств и языков четвертого поколения (4GL). Целесообразность применения CASE-средств (upper CASE), прежде всего интегрированных, определяется возможностью точного учета требований конечного пользователя к проектируемой системе, значительным снижением уровня системных ошибок в проекте до начала программирования и тем самым снижением общей трудоемкости разработки и особенно отладки программ.

5.1 CASE-системы, поддерживающие объектно-ориентированную методологию.

Одной из наиболее известных CASE систем, поддерживающей ООМ является семейство CASE-средств объектно-ориентированного анализа, проектирования и программирования Rational Rose фирмы Rational Software Corp.

Rational Rose предназначено для автоматизации этапов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose использует синтез-методологию объектно-ориентированного анализа и проектирования, основанную на подходах трех ведущих специалистов в данной области: Буча, Рамбо и Джекобсона и поддерживает универсальную нотацию для моделирования объектов UML. Конкретный вариант Rational Rose определяется языком, на котором генерируются коды программ (C++, Visual C, Java, Smalltalk, Visual Basic, PowerBuilder, Ada, и др.). Основной вариант - Rational Rose позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций, а также генерировать программные коды. Кроме того, Rational Rose содержит средства реинжиниринга программ, обеспечивающие повторное использование программных компонент в новых проектах.

Структура и функции

В основе работы Rational Rose лежит построение различного рода диаграмм и спецификаций, определяющих логическую и физическую структуры модели, ее статические и динамические аспекты. В их число входят диаграммы классов, прецедентов, состояний, последовательностей и кооперации, компонентов, процессов [21].

В составе Rational Rose можно выделить шесть основных структурных компонент: репозиторий, графический интерфейс пользователя, средства просмотра проекта (browser), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генератор кодов (индивидуальный для каждого языка) и анализатор, обеспечивающий реинжиниринг - восстановление модели проекта по исходным текстам программ.

Репозиторий представляет собой объектно-ориентированную базу данных. Средства просмотра обеспечивают "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 логической и физической моделях проекта, формируют файлы заголовков и файлы описаний классов и объектов. Создаваемый таким образом скелет программы может быть уточнен путем прямого программирования на языке C++. Анализатор создает модули проектов в форме 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 обеспечивает возможность повторного использования программных компонент.

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

диаграммы классов;

диаграммы состояний;

диаграммы взаимодействия;

диаграммы модулей;

диаграммы процессов;

спецификации классов, объектов, атрибутов и операций

заготовки текстов программ;

модель разрабатываемой программной системы.

Последний из перечисленных документов является текстовым файлом, содержащим всю необходимую информацию о проекте (в том числе необходимую для получения всех диаграмм и спецификаций).

Тексты программ являются заготовками для последующей работы программистов. Они формируются в рабочем каталоге в виде файлов типов .h (заголовки, содержащие описания классов) и .cpp (заготовки программ для методов). Система включает в программные файлы собственные комментарии, которые начинаются с последовательности символов //##. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

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 категория классов или подсистема.

Для управляемой подмодели предусмотрены операции:

- загрузка подмодели в память;
- выгрузка подмодели из памяти;
- сохранение подмодели на диске в виде отдельного файла;
- установка защиты от модификации;
- замена подмодели в памяти на новую.

Наиболее эффективно групповая работа организуется при интеграции Rational Rose со специальными средствами управления конфигурацией и контроля версий (PVCS). В этом случае защита от модификации устанавливается на все управляемые подмодели, кроме тех, которые выделены конкретному разработчику. В этом случае признак защиты от записи устанавливается для файлов, которые содержат подмодели, поэтому при считывании "чужих" подмоделей защита их от модификации сохраняется и случайные воздействия окажутся невозможными.

Rational Rose функционирует на различных платформах: IBM PC (в среде Windows), Sun SPARC stations (UNIX, Solaris, SunOS), Hewlett-Packard (HP UX), IBM RS/6000 (AIX).

СПИСОК ЛИТЕРАТУРЫ

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд,.Пер. с англ. -М.: "Издательство Бином", СПб:"Невский диалект", 1998. -560с.
2. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя: Пер. с англ. - М.:ДМК, 2000. -432с.
3. Липаев В.В., Филинов Е.Н. Мобильность программ и данных в открытых информационных системах. - М.: Научная книга, -1997. -368с.
4. Боггс У.,Боггс М. UML и Rational Rose,Пер. с англ. -М.: Издательство "ЛОРИ", 2000. -580с.
5. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. -М. Финансы и статистика, 2000. -352с

СОДЕРЖАНИЕ