**Лекция 9 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## Содержание

[Содержание 1](#_Toc280432959)

[ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ 2](#_Toc280432960)

[ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ СИСТЕМЫ 2](#_Toc280432961)

[Идентификация архитектурных решений и механизмов 2](#_Toc280432962)

[Выявление подсистем и интерфейсов 2](#_Toc280432963)

[Формирование архитектурных уровней 3](#_Toc280432964)

[Проектирование структуры потоков управления 4](#_Toc280432965)

[Проектирование конфигурации системы 4](#_Toc280432966)

[ПРОЕКТИРОВАНИЕ ЭЛЕМЕНТОВ СИСТЕМЫ 5](#_Toc280432967)

[Уточнение описания вариантов использования 5](#_Toc280432968)

[Проектирование классов 5](#_Toc280432969)

[Уточнение связей между классами. 7](#_Toc280432970)

[Проектирование баз данных 7](#_Toc280432971)

[! Следует запомнить 7](#_Toc280432972)

[S Основные понятия 8](#_Toc280432973)

[? Вопросы для самоконтроля 8](#_Toc280432974)

### ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ

Целью объектно-ориентированного проектирования является адаптация предварительного системного проекта (набора классов «анализа»), составляющего стабильную основу архитектуры системы, к среде реализации с учетом всех нефункциональных требований.

Объектно-ориентированное проектирование включает два вида деятельности:

* проектирование архитектуры системы;
* проектирование элементов системы.

### ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ СИСТЕМЫ

Проектирование архитектуры системы выполняется архитектором системы и включает в себя:

* идентификацию архитектурных решений и механизмов, необходимых для проектирования системы;
* анализ взаимодействий между классами анализа, выявление подсистем и интерфейсов;
* формирование архитектурных уровней;
* проектирование структуры потоков управления;
* проектирование конфигурации системы.

### Идентификация архитектурных решений и механизмов

В процессе проектирования определяются детали реализации архитектурных механизмов, обозначенных в процессе анализа.

Поскольку практически все механизмы — это некоторые типовые решения (образцы), они документируются в проекте (модели) с помощью кооперации со стереотипом «mechanism», при этом структурная часть механизма описывается с помощью диаграмм классов, а поведение — с помощью диаграмм взаимодействия.

### Выявление подсистем и интерфейсов

Декомпозиция системы на подсистемы реализует принцип модульности (см. подразд. 2.4.1). Целями такой декомпозиции являются:

* повышение уровня абстракции системы;
* декомпозиция системы на части, которые могут независимо:
  + конфигурироваться или поставляться;
  + разрабатываться (при условии стабильности интерфейсов);
  + размещаться в распределенной среде;
  + изменяться без воздействия на остальные части системы;
* разделение системы на части из соображений ограничения доступа к основным ресурсам;
* представление существующих продуктов или внешних систем (компонентов), которые не подлежат изменениям.

Первым действием архитектора при выявлении подсистем является преобразование классов анализа в проектные классы (design classes). По каждому классу анализа принимается одно из двух решений:

* класс анализа отображается в проектный класс, если он простой или представляет единственную логическую абстракцию;
* сложный класс анализа может быть разбит на несколько классов, преобразован в пакет или в подсистему.

Объединение классов в подсистемы осуществляется, исходя из следующих соображений:

* функциональная связь: объединяются классы, участвующие в реализации варианта использования и взаимодействующие только друг с другом;
* обязательность: совокупность классов, реализующая функциональность, которая может быть удалена из системы или заменена на альтернативную;
* связанность: объединение в подсистемы сильно связанных классов;
* распределение: объединение классов, размещенных на конкретном узле сети.
* Примеры возможных подсистем:
* совокупность классов, обеспечивающих сложный комплекс функций (например, обеспечение безопасности и защиты данных);
* граничные классы, реализующие сложный пользовательский интерфейс или интерфейс с внешними системами;
* различные продукты: коммуникационное ПО, доступ к базам данных, общие утилиты (библиотеки), различные прикладные пакеты.

При создании подсистем в модели выполняются следующие преобразования:

* •объединяемые классы помещаются в специальный пакет с именем подсистемы и стереотипом «subsystem»;
* спецификации операций классов, образующих подсистему, выносятся в интерфейс подсистемы — класс со стереотипом «Interface»;

описание интерфейса должно включать:

* имя интерфейса, отражающее его роль в системе; текстовое описание интерфейса размером в небольшой абзац, отражающее его обязанности;
* описание операций интерфейса (имя, отражающее результат операции, алгоритм выполнения операции, возвращаемое значение, параметры с их типами); характер использования операций интерфейса и порядок их выполнения документируется с помощью диаграмм взаимодействия, описывающих взаимодействие классов подсистемы при реализации операций интерфейса, которые вместе с диаграммой классов подсистемы объединяются в кооперацию с именем интерфейса и стереотипом «interface realization»;
* в подсистеме создается класс-посредник со стереотипом «subsystem proxy», управляющий реализацией операций интерфейса.

Все интерфейсы подсистем должны быть полностью определены в процессе проектирования архитектуры, поскольку они будут служить в качестве точек синхронизации при параллельной разработке системы.

### Формирование архитектурных уровней

В процессе анализа было принято предварительное решение о выделении архитектурных уровней (в случае системы регистрации — четырех уровней в соответствии с образцом «Уровни»). В процессе проектирования все элементы системы должны быть распределены по данным уровням. С точки зрения модели это означает распределение проектных классов по пакетам, соответствующим архитектурным уровням (пакетам со стереотипом «layer»). В сложной системе архитектурные уровни могут разбиваться на подуровни, количество и структура которых, как было сказано выше, зависят от сложности предметной области и среды реализации. Подуровни могут формироваться, исходя из следующих критериев:

* группировка элементов с максимальной связанностью;
* распределение в соответствии со структурой организации (обычно это касается верхних уровней архитектуры);
* распределение в соответствии с различными областями компетенции разработчиков (предметная область, сети, коммуникации, базы данных, безопасность и др.);
* группировка отдельных компонентов распределенной системы;
* распределение в соответствии с различной степенью безопасности и секретности.

### Проектирование структуры потоков управления

Проектирование структуры потоков управления выполняется при наличии в системе параллельных процессов (параллелизма). Цель проектирования — выявление существующих в системе процессов, характера их взаимодействия, создания, уничтожения и отображения в среду реализации. Требование параллелизма возникает в следующих случаях:

* необходимо распределение обработки между различными процессорами или узлами;
* система управляется потоком событий (event-driven system);
* вычисления в системе обладают высокой интенсивностью;
* в системе одновременно работает много пользователей.

Например, система регистрации курсов обладает свойством параллелизма, поскольку она должна допускать одновременную работу многих пользователей (студентов и профессоров), каждый из которых порождает в системе отдельный процесс.

Понятие ***процесс (process)*** трактуется следующим образом:

* это ресурсоемкий поток управления, который может выполняться параллельно с другими процессами;
* он выполняется в независимом адресном пространстве и в случае высокой сложности может разделяться на два ***потока*** или более;
* объект любого класса должен существовать внутри хотя бы одного процесса.

***Поток (thread)*** *—* это облегченный поток управления, который может выполняться параллельно с другими потоками в рамках одного и того же процесса в общем адресном пространстве. Необходимость создания потоков в системе регистрации курсов диктуется следующими требованиями:

* если курс окажется заполненным в то время, когда студент формирует свой учебный график, включающий данный курс, то он должен быть извещен об этом (необходим неза висимый процесс, управляющий доступом к информации конкретных курсов);
* существующая база данных каталога курсов не обеспечивает требуемую производительность (необходим процесс промежуточной обработки — подкачки данных).

Реализация процессов, и потоков обеспечивается средствами операционной системы.

Для моделирования структуры потоков управления исполь­зуются так называемые активные классы[[1]](#footnote-1) — классы со стереотипами «process» и «thread». Активный класс владеет собственным процессом или потоком и может инициировать управляющие воздействия. Связи между процессами моделируют­ся как зависимости. Потоки могут существовать только внутри процессов, поэтому связи между процессами и потоками моделируются как композиции. Модель потоков управления помещается в пакет Process View.

### Проектирование конфигурации системы

Если создаваемая система является распределенной, то необходимо спроектировать ее конфигурацию в вычислительной среде, т.е. описать вычислительные ресурсы, коммуникации между ними и использование ресурсов различными системными процессами.

Распределенная сетевая конфигурация системы моделируется с помощью диаграммы размещения.

Распределение процессов, составляющих структуру потоков управления, по узлам сети производится с учетом следующих факторов:

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

### ПРОЕКТИРОВАНИЕ ЭЛЕМЕНТОВ СИСТЕМЫ

Проектирование элементов системы выполняется проекти­ровщиками и включает в себя:

* уточнение описания вариантов использования;
* проектирование классов;
* проектирование баз данных.

### Уточнение описания вариантов использования

Уточнение описания вариантов использования заключается в модификации их диаграмм взаимодействия и диаграмм классов с учетом вновь появившихся на шаге проектирования классов и подсистем, а также проектных механизмов.

### Проектирование классов

*Проектирование классов включает следующие действия:*

* детализация проектных классов;
* уточнение операций и атрибутов;
* моделирование состояний для классов;
* уточнение связей между классами.

***Детализация проектных классов***. Каждый граничный класс преобразуется в некоторый набор классов, в зависимости от своего назначения. Это может быть набор элементов пользовательского интерфейса, зависящий от возможностей среды разработки, или набор классов, реализующий системный или аппаратный интерфейс.

Классы-сущности с учетом соображений производительности и защиты данных могут разбиваться на ряд классов. Основанием для разбиения является наличие в классе атрибутов с различной частотой использования или видимостью. Такие атрибуты, как правило, выделяются в отдельные классы.

Классы, реализующие простую передачу информации от граничных классов к сущностям, могут быть удалены. Сохраняются классы, выполняющие существенную работу по управлению потоками событий (управление транзакциями, распределенная обработка и т.д.).

Полученные в результате уточнения классы подлежат непосредственной реализации в коде системы.

***Уточнение операций и атрибутов.*** Обязанности классов, определенные в процессе анализа и документированные в виде операций «анализа», преобразуются в операции, которые будут реализованы в коде. При этом:

* каждой операции присваивается краткое имя, характеризующее ее результат;
* определяется полная сигнатура операции (в соответствии с нотацией, принятой в языке UML;
* создается краткое описание операции, включая смысл всех ее параметров;
* определяется видимость операции: public, private или protected;
* определяется область действия операции: экземпляр (операция объекта) или классификатор (операция класса);
* может быть составлено описание алгоритма выполнения операции (с использованием диаграмм деятельности в виде блок-схем, а также диаграмм взаимодействия различных объектов при выполнении операции).
* Уточнение атрибутов классов заключается в следующем:
* кроме имени атрибута, задается его тип и значение по умолчанию (необязательное);
* учитываются соглашения по именованию атрибутов, принятые в проекте и языке реализации;
* задается видимость атрибутов: public, private или protected;
* при необходимости определяются производные (вычисляемые) атрибуты.

*Моделирование состояний для классов.* Если некоторый объект всегда одинаково реагирует на событие, то он считается *независимым от состояния* по отношению к этому событию. В отличие от него *зависимые от состояния* объекты по-разному реагируют на одно и то же событие в зависимости от своего состояния. Обычно в экономических ИС содержится очень мало объектов, зависимых от состояния, а системы управления технологическими процессами (системы реального времени) зачастую содержат множество таких объектов.

Если в системе присутствуют зависимые от состояния объекты со сложной динамикой поведения, то для них можно построить модель, описывающую состояния объектов и переходы между ними. Эта модель представляется в виде диаграмм состояний.

1. *Идентификация состояний.* Признаками для выявления состояний являются изменение значений атрибутов объекта или создание и уничтожение связей с другими объектами.

2. *Идентификация событий.* События связаны, как правило, с выполнением некоторых операций.

*Идентификация переходов между состояниями.* Переходы вызываются событиями.

Историческое состояние (обозначенное на диаграмме окружностью с буквой «Н») - это псевдосостояние, которое восстанавливает предыдущее активное состояние в композитном состоянии. Оно позволяет композитному состоянию Open запоминать, какое из вложенных состояний (Unassigned или Assigned) было текущим в момент выхода из Open, для того, чтобы любой из переходов в Open (add student или remove student) возвращался именно в это вложенное состояние, а не в начальное состояние. Построение диаграмм состояний может оказать следующее воздействие на описание классов:

* события могут отображаться в операции класса;
* особенности конкретных состояний могут повлиять на детали выполнения операций;
* описание состояний и переходов может помочь при определении атрибутов класса.

### Уточнение связей между классами.

В процессе проектирования связи между классами (ассоциации, агрегации и обобщения) подлежат уточнению.

* Ассоциации между граничными и управляющими классами отражают связи, динамически возникающие между соответствующими объектами в потоке управления. Для таких связей достаточно обеспечить видимость классов, поэтому они преобразуются в зависимости.
* Если для некоторых ассоциаций нет необходимости в двунаправленной связи, то вводятся направления навигации.
* Агрегации, обладающие свойствами композиции (см. подразд. 2.4.2), преобразуются в связи композиции.

### Проектирование баз данных

Проектирование БД зависит от типа используемой для хранения данных СУБД - объектной или реляционной. Для объектных БД никакого проектирования не требуется, поскольку классы-сущности непосредственно отображаются в БД. Для реляционных БД классы-сущности объектной модели должны быть отображены в таблицы реляционной БД. Совокупность таблиц и связей между ними может быть представлена в виде диаграммы классов, которая по существу является ER-диаграммой. Набор правил, применяемых при отображении классов в таблицы БД, фактически совпадает с правилами преобразования сущностей и связей, описанными в подразд. 4.1. В технологии RUP, в частности, для такого отображения используется специальный инструмент - Data Modeler. Он выполняет преобразование классов-сущностей в классы-таблицы с последующей генерацией описания БД на SQL.

Для описания схемы БД применяется следующий набор элементов языка UML со своими стереотипами (профиль UML):

* таблица представляется в виде класса со стереотипом «Table»;
* представление изображается в виде класса со стереотипом «View»;
* столбец таблицы представляется в виде атрибута класса с соответствующим типом данных;
* обычная ассоциация и агрегация представляются в виде ассоциации со стереотипом «Non-Identifying» (в терминологии IDEF1X — неидентифицирующей связи);
* композиция представляется в виде ассоциации со стереотипом «Identifying» (в терминологии IDEF1X - идентифицирующей связи);
* схема БД представляется в виде пакета со стереотипом «Schema», содержащего классы-таблицы;
* контейнер хранимых процедур представляется в виде класса со стереотипом «SP Container»;
* ограничения целостности, индексы и триггеры представляются в виде операций классов-таблиц со стереотипами «РК» (Primary key), «FK» (Foreign key), «Unique», «Check», «Index» и «Trigger»;
* физическая база данных представляется в виде компонента со стереотипом «Database».

## ! Следует запомнить

Целью объектно-ориентированного анализа является трансформация функциональных требований к ПО в предварительный системный проект и создание стабильной основы архитектуры системы. В процессе проектирования системный проект «погружается» в среду реализации с учетом всех нефункциональных требований.

## S Основные понятия

Архитектурные механизмы, архитектурные уровни, классы анализа, проектные классы, процесс, поток.

## ? Вопросы для самоконтроля

1. Перечислите правила формирования схемы базы данных из ERM.
2. К каким последствиям для системы может привести отсутствие архитектурного анализа?
3. Что дает использование образцов распределения обязанностей?
4. Какие классы нуждаются в моделировании состояний?
5. Охарактеризуйте достоинства и область применения методики анализа и проектирования Rational Unified Process.

1. *Буч Г. и др.* Язык UML. Руководство пользователя.: Пер. с англ. / Г. Буч, Дж. Рамбо, А. Джекобсон. - М.: ДМК, 2000. [↑](#footnote-ref-1)