**Лекция 8 ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ**

Содержание

[1. АРХИТЕКТУРНЫЙ АНАЛИЗ 2](#_Toc416503448)

[утверждение общих стандартов (соглашений) моделирования и документирования системы; 2](#_Toc416503449)

[предварительное выявление архитектурных механизмов (механизмов анализа); 2](#_Toc416503450)

[формирование набора основных абстракций предметной области (классов анализа); 2](#_Toc416503451)

[формирование начального представления архитектурных уровней. 2](#_Toc416503452)

[2. АНАЛИЗ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ 4](#_Toc416503453)

[Идентификация классов, участвующих в реализации потоков событий варианта использования 5](#_Toc416503454)

[Распределение поведения, реализуемого вариантом использования, между классами 6](#_Toc416503455)

[Определение атрибутов и ассоциаций классов 13](#_Toc416503456)

[Унификация классов анализа 14](#_Toc416503457)

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.

Объектно-ориентированный анализ включает два вида деятельности:

* архитектурный анализ и
* анализ вариантов использования.

## 1. АРХИТЕКТУРНЫЙ АНАЛИЗ

Архитектурный анализ выполняется архитектором системы и включает в себя:

#### утверждение общих стандартов (соглашений) моделирования и документирования системы;

#### предварительное выявление архитектурных механизмов (механизмов анализа);

#### формирование набора основных абстракций предметной области (классов анализа);

#### формирование начального представления архитектурных уровней.

***Соглашения моделирования*** определяют:

* используемые диаграммы и элементы модели;
* правила их применения;
* соглашения по именованию элементов модели;
* организацию модели (пакеты).

***Пример набора соглашений моделирования:***

* Имена вариантов использования должны быть короткими глагольными фразами.
* Имена классов должны быть существительными, соответствующими, по возможности, понятиям предметной области.
* Имена классов должны начинаться с заглавной буквы.
* Имена атрибутов и операций должны начинаться со строчной буквы.
* Составные имена должны быть сплошными, без подчеркиваний, каждое отдельное слово должно начинаться с заглавной буквы.

Все классы и диаграммы, описывающие предварительный системный проект, помещаются в пакет с именем Analysis Model.

***Архитектурные механизмы*** отражают нефункциональные требования к системе (надежность, безопасность, хранение данных в конкретной среде, интерфейсы с внешними системами и т.д.) и их реализацию в архитектуре системы. В процессе анализа они только обозначаются, отвлекаясь от особенностей среды реализации, а все детали их реализации определяются в процессе проектирования.

***Архитектурные механизмы***, по существу, представляют собой набор типовых решений, или образцов, принятых в качестве стандарта данного проекта.

***Идентификация основных абстракций*** заключается в предварительном определении набора классов системы (классов анализа) на основе описания предметной области и спецификации требований к системе (в частности, глоссария). Способы идентификации основных абстракций аналогичны способам идентификации сущностей в модели ERM.

Так, для системы регистрации идентифицировано пять классов анализа (рисунок 8.1):

* Студент;
* Профессор;
* Учебный график;
* Курс;
* Предлагаемый курс.

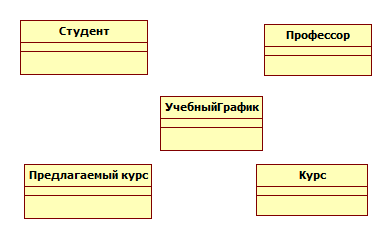


Рисунок 8.1 - Система регистрации

***Архитектурные уровни*** образуют иерархию уровней представления любой крупномасштабной системы. В практике разработки таких систем существует ряд типовых решений — архитектурных образцов, среди которых наиболее распространенным является образец «Уровни» (Layers). «Уровни» – способ декомпозиции приложения на набор слоев, соответствующих различным уровням абстракции. Этот образец можно описать следующим образом:

*Наименование образца:*

«Уровни».

*Контекст:*

Крупномасштабные системы, нуждающиеся в декомпозиции.

*Проблема:*

Архитектура крупномасштабной системы должна удовлетворять следующим требованиям:

* компоненты системы должны иметь возможность замены;
* изменения в одних компонентах не должны сильно затрагивать другие;
* однородные функции должны группироваться вместе;
* размер компонентов не должен быть слишком большим.

*Решение:*

Предлагается базовый вариант, включающий следующие уровни (сверху вниз) (рисунок 8.2):

* прикладной (Application Subsystems) - набор компонентов, реализующих основную функциональность системы, отраженную в вариантах использования;
* бизнес-уровень (Business-specific) - набор компонентов, специфичных для конкретной предметной области;
* промежуточный (Middleware) - различные платформо-независимые сервисы (библиотеки пользовательского интерфейса, брокеры запросов и др.);
* системный (System software) - ПО для вычислительной и сетевой инфраструктуры (ОС, сетевые протоколы и др.).

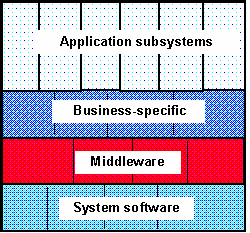


Рисунок 8.2 - Иерархия уровней представления крупномасштабной системы

Благодаря использованию этого образца система получается модифицируемой (т. е. внесение в нее изменений сравнительно не трудоемко) и мобильной (т. е. может переноситься на другие платформы)

Архитектурные уровни представляются в модели в виде пакетов со стереотипом «layer». Количество и структура уровней зависят от сложности предметной области и среды реализации. В рамках архитектурного анализа определяется начальная структура модели (набор пакетов и их зависимостей) и рассматриваются только верхние уровни (прикладной и бизнес-уровень).

## 2. АНАЛИЗ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Анализ вариантов использования выполняется проектировщиками и включает в себя:

* идентификацию классов, участвующих в реализации потоков событий варианта использования;
* распределение поведения, реализуемого вариантом использования, между классами (определение обязанностей классов);
* определение атрибутов и ассоциаций классов;
* унификацию классов анализа.

#### Идентификация классов, участвующих в реализации потоков событий варианта использования

В потоках событий варианта использования выявляются классы трех типов.

***Граничные классы (Boundary)*** — служат посредниками при взаимодействии внешних объектов с системой. Как правило, для каждой пары «действующее лицо — вариант использования» определяется один граничный класс. Типы граничных классов: пользовательский интерфейс (обмен информацией с пользователем без деталей интерфейса — кнопок, списков, окон), системный интерфейс и аппаратный интерфейс (используемые протоколы без деталей их реализации).

***Классы-сущности (Entity)*** — представляют собой основные абстракции (понятия) разрабатываемой системы, рассматриваемые в рамках конкретного варианта использования. Источники выявления классов-сущностей: основные абстракции, созданные в процессе архитектурного анализа, глоссарий, описание потоков событий вариантов использования, сущности, описанные в модели бизнес-анализа (при наличии бизнес-модели).

***Управляющие классы (Control)*** — обеспечивают координацию поведения объектов в системе. Они могут отсутствовать в некоторых вариантах использования, ограничивающихся простыми манипуляциями с хранимыми данными. Как правило, для каждого варианта использования определяется один управляющий класс. Примеры управляющих классов: менеджер транзакций, координатор ресурсов, обработчик ошибок.

Классы анализа отражают функциональные требования к системе и моделируют объекты предметной области. Совокупность классов анализа представляет собой начальную концептуальную модель системы.

Пример набора классов, участвующих в реализации варианта использования «Зарегистрироваться на курсы», приведен на рисунке 8.3.

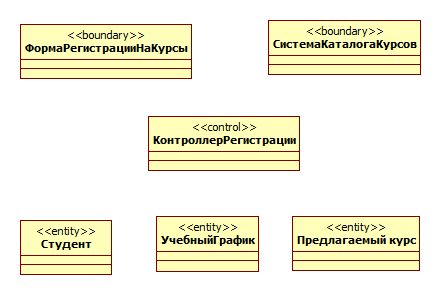


Рисунок 8.3 - Пример набора классов

Концептуальная [модель](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)&action=edit&redlink=1) — модель предметной области, состоящей из перечня взаимосвязанных [понятий](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5), используемых для описания этой области, вместе со [свойствами](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE) и характеристиками, [классификацией](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) этих понятий, по типам, ситуациям, признакам в данной области и [законов](https://ru.wikipedia.org/w/index.php?title=%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%BF%D1%80%D0%B8%D1%80%D0%BE%D0%B4%D1%8B&action=edit&redlink=1) протекания процессов в ней.

#### Распределение поведения, реализуемого вариантом использования, между классами

Квалифицированное распределение обязанностей между классами является наиболее важной частью объектно-ориентированного анализа. Исходя из назначения трех выделенных типов классов, можно кратко охарактеризовать распределение обязанностей между ними:

* граничные классы отвечают за взаимодействие с внешней средой системы (действующими лицами);
* классы-сущности отвечают за хранение и манипулирование данными;
* Управляющие классы координируют потоки событий варианта использования.

Более детальное распределение обязанностей (в виде операций классов) выполняется с помощью диаграмм взаимодействия (диаграмм последовательности и кооперативных диаграмм).

В процессе анализа конкретного варианта использования в первую очередь строится диаграмма последовательности (одна или более), описывающая основной поток событий и его подчиненные потоки. Для каждого альтернативного потока событий строится отдельная диаграмма (из соображений простоты наглядного восприятия). Нецелесообразно описывать тривиальные потоки событий (например, в потоке участвует только один объект).

На рисунке 8.4 приведена диаграмма последовательности «Зарегистрироваться на курсы» для основного потока событий.

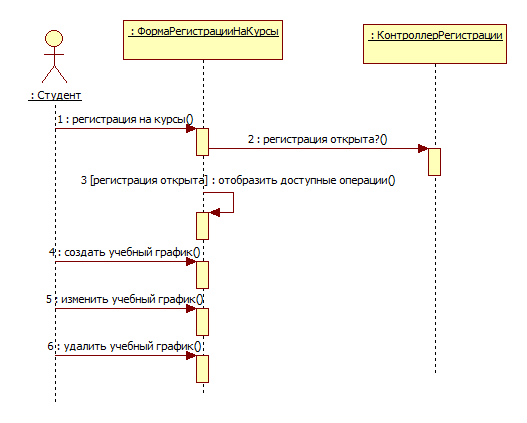


Рисунок 8.4 - Диаграмма последовательности «Зарегистрироваться на курсы»

Обязанности каждого класса определяются исходя из сообщений на диаграммах взаимодействия и документируются в классах в виде операций «анализа», которые появляются там в процессе построения диаграмм взаимодействия (соотнесения сообщений с операциями). Каждая операция «анализа» класса соответствует некоторому сообщению, принимаемому объектами данного класса. В процессе проектирования каждая операция «анализа» преобразуется в одну или более операций класса, которые в дальнейшем будут реализованы в коде системы.

После построения диаграммы последовательности «Зарегистрироваться на курсы» варианта использования примет следующий вид (рисунок 8.5).

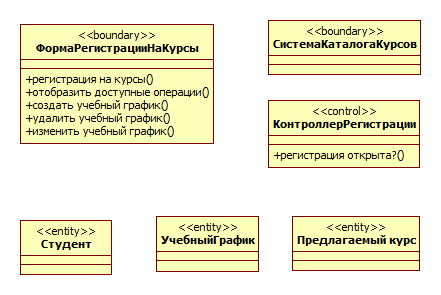


Рисунок 8.5 – Измененная диаграмма последовательности «Зарегистрироваться на курсы»

При построении диаграмм взаимодействия возникают проблемы правильного распределения обязанностей между классами. Для их решения существует ряд образцов[[1]](#footnote-1), некоторые из них приведены ниже.

**Образец «Information Expert»**

*Проблема:*

Необходимо определить наиболее общий принцип распределения обязанностей между классами. В системе могут быть определены сотни классов, выполняющие тысячи обязанностей. При правильном их распределении система становится гораздо проще для понимания, сопровождения и развития. Кроме того, появляется возможность повторного использования уже разработанных компонентов в последующих приложениях.

*Решение:*

Следует назначить обязанность информационному эксперту — классу, у которого имеется информация, требуемая для выполнения обязанности.

*Следствия:*

При распределении обязанностей образец Information Expert используется гораздо чаще любого другого образца. Большинство сообщений на приведенных выше диаграммах взаимодействия соответствуют данному образцу. В нем определены основные принципы, которые уже давно используются в объектно-ориентированном анализе и проектировании. Образец Information Expert не содержит неясных или запутанных идей и отражает обычный интуитивно понятный подход. ***Он заключается в том, что объекты осуществляют действия, связанные с имеющейся у них информацией. Если информация распределена между различными объектами, то при выполнении общей задачи они должны взаимодействовать с помощью сообщений.***

Образец Information Expert, как и многие другие понятия объектной технологии, имеет соответствующую аналогию в реальном мире. Например, в организации обязанности обычно распределяются между теми служащими, у которых имеется необходимая для выполнения поставленной задачи информация. Ответственность за создание отчета о прибыли и убытках должен нести тот из служащих, кто имеет доступ ко всей информации, необходимой для создания такого отчета. Возможно, лучше всего для выполнения этой обязанности подойдет руководитель финансового отдела предприятия. Программные объекты взаимодействуют между собой и обмениваются информацией так же, как люди. Начальник финансового отдела компании может запросить требуемые данные у бухгалтеров, работающих со счетами по дебиторской и кредиторской задолженности, чтобы составить отдельные отчеты по кредиту и дебету.

В некоторых ситуациях применение образца Information Expert нежелательно. Например, в системе регистрации нужно определить, какой объект должен отвечать за сохранение информации об учебных курсах в базе данных. Информация, подлежащая сохранению, «известна» объекту Курс, а значит, согласно образцу Information Expert, на класс Курс следует возложить обязанность по сохранению. Логическим следствием такого рассуждения является вывод о том, что каждый объект должен отвечать за сохранение себя в базе данных. Однако при этом возникает проблема перегруженности класса лишними обязанностями, поскольку класс Курс должен содержать методы обращения к базе данных, т.е. должен быть связан с вызовом операторов языка SQL или сервисов JDBC (Java Database Connectivity). Тогда этот класс не будет относиться к предметной области и моделировать учебные курсы. Кроме того, подобные действия будут дублироваться во многих других классах, информация которых подлежит постоянному хранению.

Все эти проблемы приводят к нарушению основного архитектурного принципа проектирования с разделением основных функций системы на уровни, отраженного в образце «Уровни». Разнородные функции не должны реализовываться в одном и том же компоненте. С этой точки зрения класс Курс не должен отвечать за сохранение информации в базе данных.

Основное достоинство образца Information Expert — поддержка, инкапсуляции. Для выполнения требуемых задач объекты используют собственные данные. Необходимое поведение системы обеспечивается несколькими классами, содержащими требуемую информацию. Это приводит к определениям классов, которые гораздо проще понимать и поддерживать.

**Образец «Creator»**

*Проблема:*

Нужно определить, кто должен отвечать за создание нового экземпляра некоторого класса. Создание новых объектов в объектно-ориентированной системе является одним из стандартных видов деятельности. Следовательно, при назначении обязанностей, связанных с созданием объектов, полезно руководствоваться некоторым основным принципом.

*Решение:*

Следует назначить классу В обязанность создавать экземпляры класса А, если выполняется одно из следующих условий:

* класс В агрегирует, содержит или активно использует объекты класса А;
* класс В обладает данными инициализации, которые будут передаваться объектам класса А при их создании (т.е. класс В является информационным экспертом).

Класс В при этом определяется как создатель (creator) объектов класса А.

Если несколько классов удовлетворяют этим условиям, то предпочтительнее использовать в качестве создателя класс, агрегирующий или содержащий класс А.

*Следствия:*

Образец «Creator» .определяет способ распределения обязанностей, связанный с процессом создания объектов. В объектно-ориентированных системах эта задача является одной из наиболее распространенных. Основным назначением образца Creator является выявление объекта-создателя, который при возникновении любого события должен быть связан со всеми созданными им объектами. При таком подходе обеспечивается низкая степень связанности объектов.

В некоторых случаях в качестве создателя выбирается класс, который содержит данные инициализации, передаваемые объекту во время его создания. На самом деле это пример использования образца Information Expert. В процессе создания инициализирующие данные передаются с помощью метода инициализации некоторого вида, такого, как конструктор языка программирования.

В некоторых сложных случаях вместо данного образца предпочтительнее использовать известный образец Factory (Фабрика)1[[2]](#footnote-2) и делегировать обязанность создания объектов вспомогательному классу.

**Образец «Low Coupling»**

*Проблема:*

Нужно распределить обязанности между классами таким образом, чтобы снизить взаимное влияние изменений в них и повысить возможность повторного использования.

*Решение:*

Следует распределить обязанности таким образом, чтобы обеспечить слабую связанность. Связанность (coupling) — это мера, определяющая насколько жестко один элемент связан с другими элементами, или каким количеством данных о других элементах он обладает. Элемент со слабой связанностью зависит от небольшого числа других элементов. Класс с сильной связанностью зависит от множества других классов. Наличие таких классов нежелательно, поскольку оно приводит к возникновению следующих проблем.

* Изменения в связанных классах приводят к локальным изменениям в данном классе.
* Затрудняется понимание каждого класса в отдельности.
* Усложняется повторное использование, поскольку для этого требуется дополнительный анализ классов, с которыми связан данный класс.

*Следствия:*

Образец Low Coupling поддерживает независимость классов, что, в свою очередь, повышает возможности повторного использования и обеспечивает более высокую эффективность приложения. Его нельзя рассматривать изолированно от других образцов, таких как Information Expert и High Cohesion. Он также обеспечивает выполнение одного из основных принципов проектирования, применяемых при распределении обязанностей.

Не существует абсолютной меры для определения слишком сильной связанности. Важно лишь понимать связанность объектов и не упустить тот момент, когда дальнейшее повышение связанности может привести к возникновению проблем. В целом следует руководствоваться таким принципом: классы, которые являются достаточно общими по своей природе и с высокой вероятностью будут повторно использоваться в дальнейшем, должны иметь минимальную связанность с другими классами.

Крайним случаем при реализации образца Low Coupling является полное отсутствие связывания между классами. Такая ситуация тоже нежелательна, поскольку основная идея объектного подхода выражается в системе связанных объектов, которые «общаются» между собой посредством передачи сообщений. При слишком частом использовании принципа слабого связывания система будет состоять из нескольких изолированных сложных активных объектов, самостоятельно выполняющих все операции, и множества пассивных объектов, основная функция которых сводится к хранению данных. Поэтому при создании объектно-ориентированной системы должна присутствовать некоторая оптимальная степень связывания между объектами, позволяющая выполнять основные функции посредством взаимодействия этих объектов.

Не следует применять данный образец, когда создаются связи с устойчивыми элементами. Сильная связанность с такими элементами не представляет проблемы. Например, приложение Java 2 Enterprise Edition можно жестко связать с библиотеками Java, поскольку они достаточно стабильны.

Сильная связанность сама по себе не является проблемой. Проблемой является жесткое связывание с неустойчивыми элементами. Важно понимать, что без убедительной мотивации не следует во что бы то ни стало бороться за уменьшение связанности объектов.

**Образец «High Cohesion»**

*Проблема:*

Нужно распределить обязанности между классами таким образом, чтобы каждый класс не выполнял много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к возникновению таких же проблем, как у классов с сильной связанностью.

*Решение:*

Следует распределить обязанности таким образом, чтобы обеспечить сильное сцепление. В терминах объектно-ориентированного проектирования *сцепление* (cohesion) — это мера связанности и непротиворечивости обязанностей класса. Считается, что элемент обладает сильным сцеплением, если его обязанности тесно связаны между собой, и он не выполняет излишнего объема работы. В роли таких элементов могут выступать классы, подсистемы, модули и т.д.

Классы со слабым сцеплением, как правило, выполняют обязанности, которые можно легко распределить между другими классами.

*Следствия:*

Как правило, класс с сильным сцеплением содержит сравнительно небольшое число методов, которые функционально тесно связаны между собой, и не выполняет слишком много функций. Он взаимодействует с другими классами для выполнения более сложных задач. Высокая степень однотипной функциональности в сочетании с небольшим числом операций упрощают поддержку и модификацию класса, а также возможность его повторного использования.

Образец High Cohesion, как и другие понятия объектно-ориентированной технологии проектирования, имеет аналогию в реальном мире. Известно, что человек, выполняющий большое число разнородных обязанностей (особенно тех, которые можно легко распределить между другими людьми), работает не очень эффективно. Особенно это касается менеджеров, которые не умеют распределять обязанности между своими подчиненными.

***Слабая связанность и сильное сцепление*** — основные принципы проектирования ПО. Объектное проектирование полностью с ними согласуется, поскольку одним из его базовых принципов является модульность.

Однако существуют ситуации, когда слабое сцепление оказывается оправданным. Одна из таких ситуаций возникает в том случае, когда обязанности или код группируются в одном классе или компоненте для упрощения его поддержки одним человеком. Другой пример слабого сцепления имеет отношение к распределенным серверным объектам. Поскольку быстродействие системы определяется производительностью удаленных объектов и их взаимодействием, иногда желательно создать несколько крупных серверных объектов со слабым сцеплением, предоставляющих интерфейс многим операциям. Это приведет к уменьшению числа удаленных вызовов и, как следствие, к повышению производительности.

Набор обязанностей классов, полученный в результате их распределения, должен быть проанализирован на предмет выявления и устранения следующих проблем:

* дублирования одинаковых обязанностей в различных классах;
* противоречивых обязанностей в рамках класса;
* классов с одной обязанностью или вообще без обязанностей;
* классов, взаимодействующих с большим количеством других классов.

#### Определение атрибутов и ассоциаций классов

Атрибуты классов анализа определяются, исходя из знаний о предметной области, требований к системе, глоссария и бизнес-модели. В процессе анализа атрибуты определяются только для классов-сущностей, и они имеют тот же смысл, что и в модели ERM. Атрибуты должны быть простыми, т.е. выразимыми с помощью простых типов данных (одной из распространенных ошибок является моделирование сложного понятия предметной области в виде атрибута). Так, после определения атрибутов для классов-сущностей, показанных на рисунке 8.5, они примут следующий вид (рисунок 8.6).

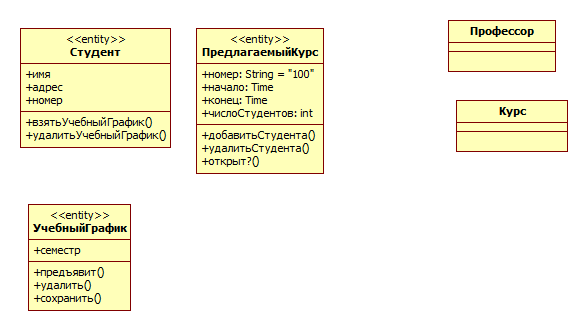


Рисунок 8.6 - Атрибуты классов анализа

Связи между классами (ассоциации) определяются в два этапа:

1. Начальный набор связей определяется на основе анализа кооперативных диаграмм. Если два объекта взаимодействуют (обмениваются сообщениями), между ними на кооперативной диаграмме должна существовать связь (путь взаимодействия), которая преобразуется в двунаправленную ассоциацию между соответствующими классами. Если сообщения между некоторой парой объектов передаются только в одном направлении, то для соответствующей ассоциации вводится направление навигации.
2. Анализируются и уточняются ассоциации между классами-сущностями. Задаются мощности ассоциаций, могут использоваться множественные ассоциации, агрегации, обобщения и ассоциации-классы.

Следует избегать использования избыточных ассоциаций и сосредоточиться на тех ассоциациях, для которых данные о связи должны сохраняться в течение некоторого времени. Это достаточно важный момент. Если модель содержит N различных классов анализа, то между ними можно установить N\*(N-1) ассоциацию, большинство из которых будет просто создавать «визуальный шум» и ухудшать наглядность диаграмм. Поэтому при добавлении ассоциаций нужно придерживаться принципа минимализма.

На рисунке 8.7 показана полная диаграмма классов варианта использования «Зарегистрироваться на курсы» (без атрибутов и операций). Ассоциации между граничными и управляющими классами, а также между управляющими классами и классами сущностями введены на основе анализа кооперативных диаграмм и в отличие от устойчивых структурных (семантических) связей между сущностями отражают связи, динамически возникающие между соответствующими объектами в потоке управления (в процессе работы приложения). Поскольку для ассоциаций это не свойственно (в процессе проектирования) они могут быть преобразованы в зависимости.

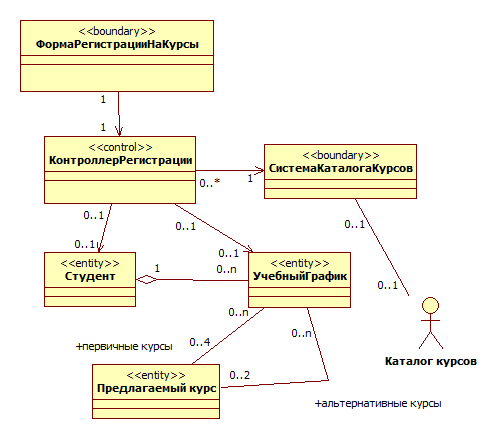


Рисунок 8.7 - Полная диаграмма классов варианта использования «Зарегистрироваться на курсы»

#### Унификация классов анализа

Унификация классов анализа заключается в проверке созданных классов на предмет выполнения следующих условий:

* имя и описание каждого класса «анализа» должно отражать сущность его роли в системе. Имена должны быть уникальными;
* классы с одинаковым поведением, или представляющие одно и то же явление, должны объединяться;
* классы-сущности с одинаковыми атрибутами должны объединяться (даже если их поведение отличается). Объединенный класс будет обладать общим поведением.

При обновлении классов должны, при необходимости, обновляться описания вариантов использования.

1. *Ларман К.* Применение UML и шаблонов проектирования. — 2-е изд.: Пер. с англ. — М.: Вильяме, 2002. [↑](#footnote-ref-1)
2. Приемы объектно-ориентированного проектирования. Паттерны про­ектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес: Пер. с англ. -СПб.: Питер, 2001. [↑](#footnote-ref-2)