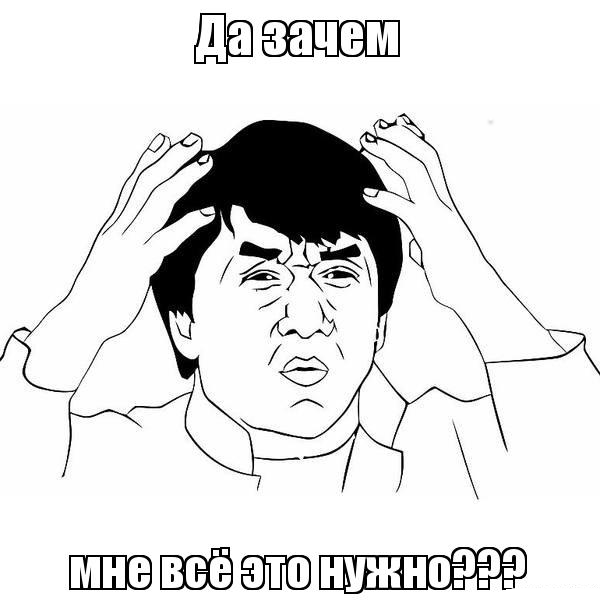
Зачем нам UML? Или как сохранить себе нервы и время

[Программирование\*](https://habr.com/ru/hub/programming/)[Анализ и проектирование систем\*](https://habr.com/ru/hub/analysis_design/)[UML Design\*](https://habr.com/ru/hub/uml/)

Из песочницы

Многие программисты, столкнувшись со сложной задачей, пренебрегают этапом проектирования, ссылаясь на то, что проектирование — это потеря времени, и в данном случае оно будет мне только мешать.



Зачастую это утверждение оказывается верным, если задача и правда небольшая и квалификации программиста достаточно для определения наиболее оптимального решения.  
  
Программисты, не использующие UML, делятся на несколько групп:

* начну писать код, а в процессе пойму, что да как;
* почитаю форумы, хабр, medium, stack overflow, книгу, записи на стенах, знаки свыше…;
* поспрашиваю у коллег, может, кто-то знает, как решить подобную задачу;
* начну рисовать квадратики и схематично покажу, какое видение задачи сформировалось у меня в сознании.

Но при решении более сложных задач заблаговременное планирование и моделирование значительно упрощают программирование. Кроме того, вносить изменения в диаграммы классов легче, чем в исходный код.  
  
Можно провести аналогию с постройкой дома. Когда кто-то хочет построить дом, он не просто бьет молотком и приступает к работе. Ему нужно иметь план — план проектирования, чтобы он мог анализировать и модифицировать свою систему.  
  
Если вы уже начали описывать на бумаге вашу задачу, это уже огромный плюс.

Что такое UML

Официальное определение из википедии.

**UML** – унифицированный язык моделирования (Unified Modeling Language) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

Проще говоря, если посмотреть картинки в поисковых системах, то станет понятно, что UML – это что-то про схемы, стрелочки и квадратики.  
  
Важно, что UML переводится как Unified Modeling Language. Главное здесь слово Unified. То есть наши картинки поймём не только мы, но и остальные, знающие UML. Получается, это такой международный язык рисования схем.

Плюсы и минусы UML проектирования

**Минусы:**

* трата времени;
* необходимость знания различных диаграмм и их нотаций.

**Плюсы:**

* возможность посмотреть на задачу с разных точек зрения;
* другим программистам легче понять суть задачи и способ ее реализации;
* диаграммы сравнительно просты для чтения после достаточно быстрого ознакомления с их синтаксисом.

Для того, чтобы разобраться, нужно ли именно вам использовать UML, необходимо рассмотреть основные диаграммы. Благодаря им складывается общая картина, дающая представление о возможностях выражения архитектурных идей в рамках бизнес-задач.  
  
Все представленные ниже диаграммы связаны между собой. Комбинируя их, мы можем добиться необходимого уровня декомпозиции отдельно взятых задач.  
  
Предлагаю познакомиться с одними из самых полезных и часто используемых диаграмм.  
Речь пойдет о диаграммах последовательности, состояний, деятельности и самой сложной из них — диаграмме классов.

Сначала я <…>, а потом <…>, а потом… Диаграмма последовательностей

Представьте, что вам нужно описать последовательность действий для заказа товара в интернет-магазине. Кто должен участвовать в процессе? Какие фазы проходит заказ прежде, чем он будет оформлен?  
  
Обычно, мы пишем длинный список этапов, которые должна пройти заявка, чтобы получить гордый статус «Оформлена». Затем описываем, кто именно будет выполнять конкретное действие. И только после этого начинаем программировать.  
  
В чем недостаток данного подхода? Он не нагляден.  
  
Представьте, перед вами лежит длинный список описанных ранее этапов и комментариев к ним. Насколько просто вам будет разобраться в нем? Сколько времени может на это потребоваться? Предполагаю, что достаточно.  
  
Альтернативой данному подходу является использование диаграммы последовательностей, представленной на рисунке ниже.

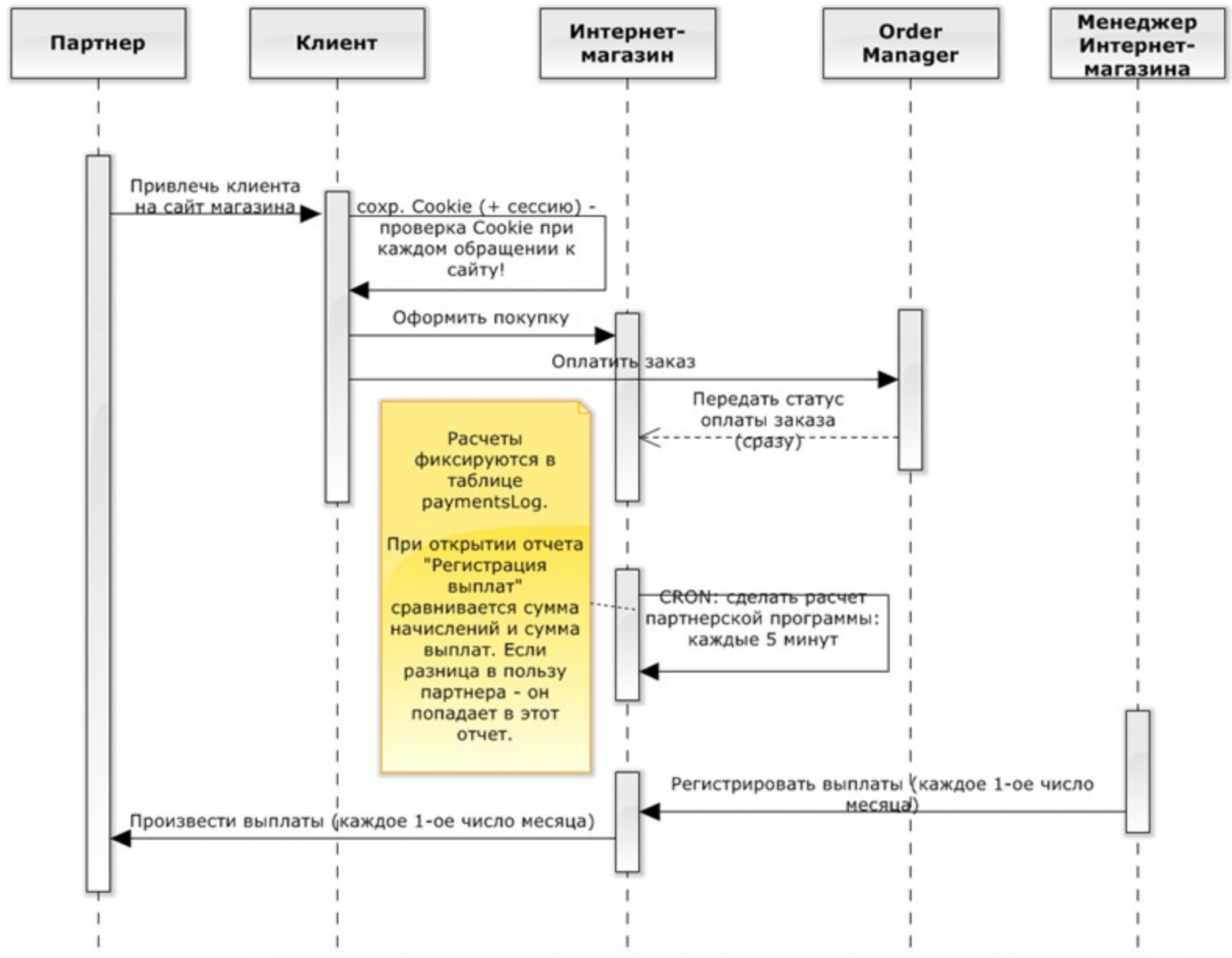


Диаграмма последовательности  
  
Сверху отображены действующие лица, а каждая стрелка это конкретное действие, связанное с ними. Подробнее о данной диаграмме можно узнать [здесь](http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl8/gl8.html)

Диаграмма состояний. Настраиваем старые электронные часы

Диаграмма состояний позволяет описать поведение отдельно взятого объекта при определенных условиях. Также она покажет нам все возможные состояния, в которых может находиться объект, а также процесс смены состояний в результате внешнего влияния.  
  
Предположим, мы программируем советские электронные часы.



Для настройки нам дано всего несколько кнопок. Довольно негусто. При этом мы знаем, что одна из кнопок переключает режим настройки часов. Другая кнопка в первом режиме меняет минуты, а во втором часы.    
  
Инструкция по настройке и так достаточно небольшая, но благодаря диаграмме состояний она визуально воспринимается гораздо проще.

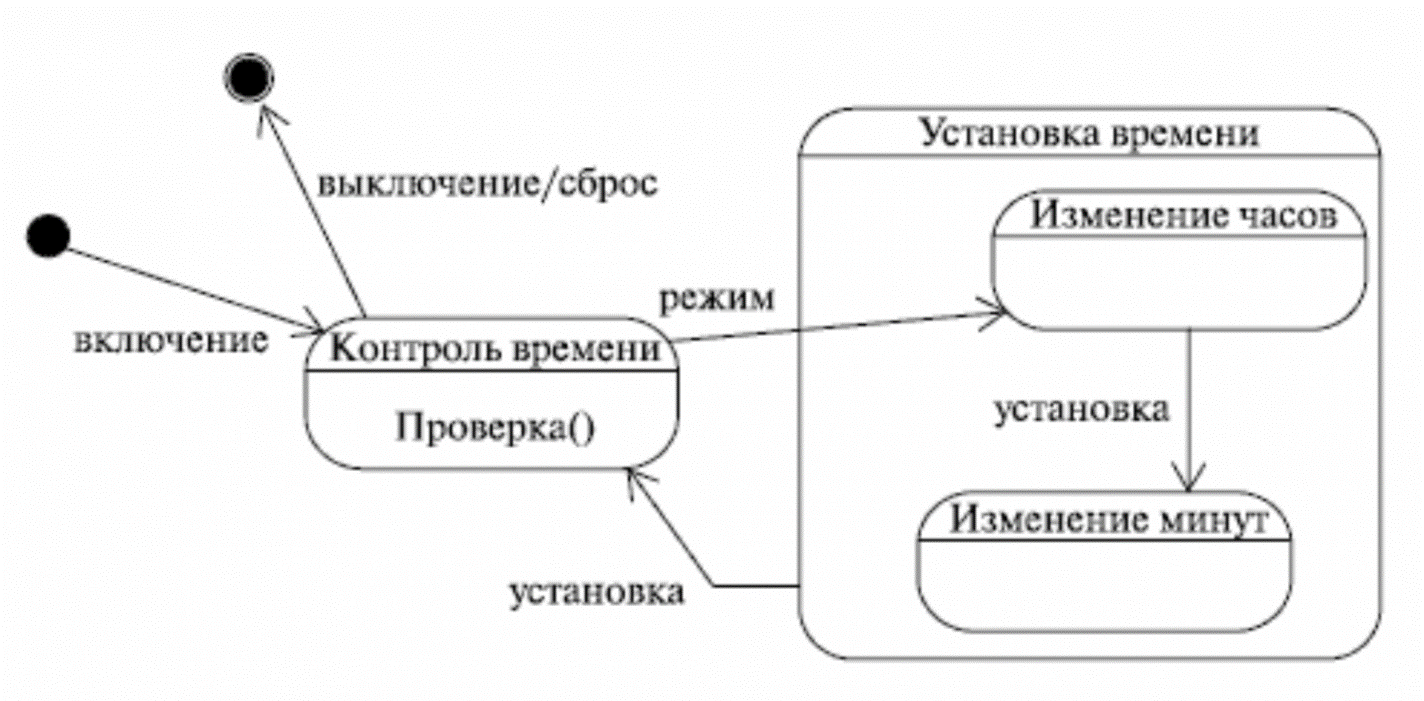
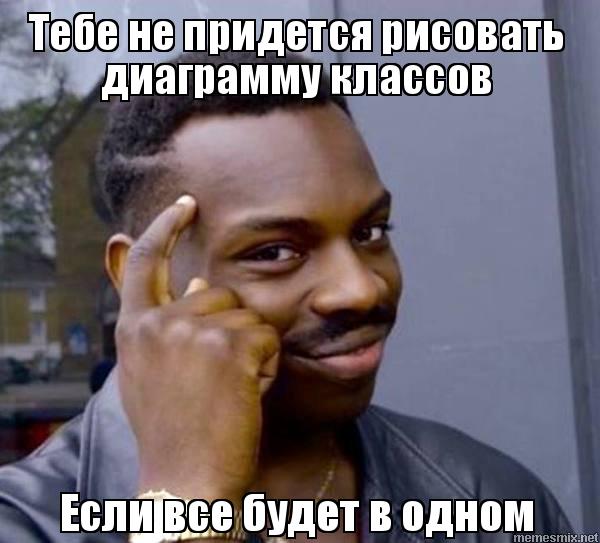


Диаграмма состояний  
  
Подробнее о диаграмме состояний можно прочитать [здесь](https://www.intuit.ru/studies/professional_skill_improvements/1364/courses/229/lecture/5954?page=4).

Диаграмма классов, или как рассказать о своем коде без кода

Диаграммы классов используются при моделировании ПС наиболее часто. Они являются одной из форм статического описания системы с точки зрения ее проектирования. Диаграмма классов не отображает динамическое поведение объектов изображенных на ней классов. На диаграммах классов показываются классы, интерфейсы и отношения между ними.

В различных документациях, описании паттернов проектирования, а также, читая хабр, все мы часто встречаем диаграмму классов. Почему же ее так часто используют?



Предположим, вам нужно спроектировать систему. Прежде чем приступить к реализации нескольких классов, вы захотите иметь концептуальное понимание системы, — какие классы мне нужны? Какая функциональность и информация будет у этих классов? Как они взаимодействуют друг с другом? Кто может видеть эти классы? И так далее.  
  
Вот где появляются диаграммы классов. Диаграммы классов — это отличный способ визуализировать классы в вашей системе, прежде чем вы начнете их кодировать. Они представляют собой статическое представление структуры вашей системы.  
  
Именно диаграмма классов дает нам наиболее полное и развернутое представление о структуре и связях в программном коде. Понимание принципов построения данной диаграммы позволяет кратко и прозрачно выражать свои мысли и идеи.  
  
Рассмотрим, как с помощью диаграммы классов описать известный паттерн проектирования «Посетитель».

«Посетитель» — это поведенческий паттерн проектирования, который позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться.

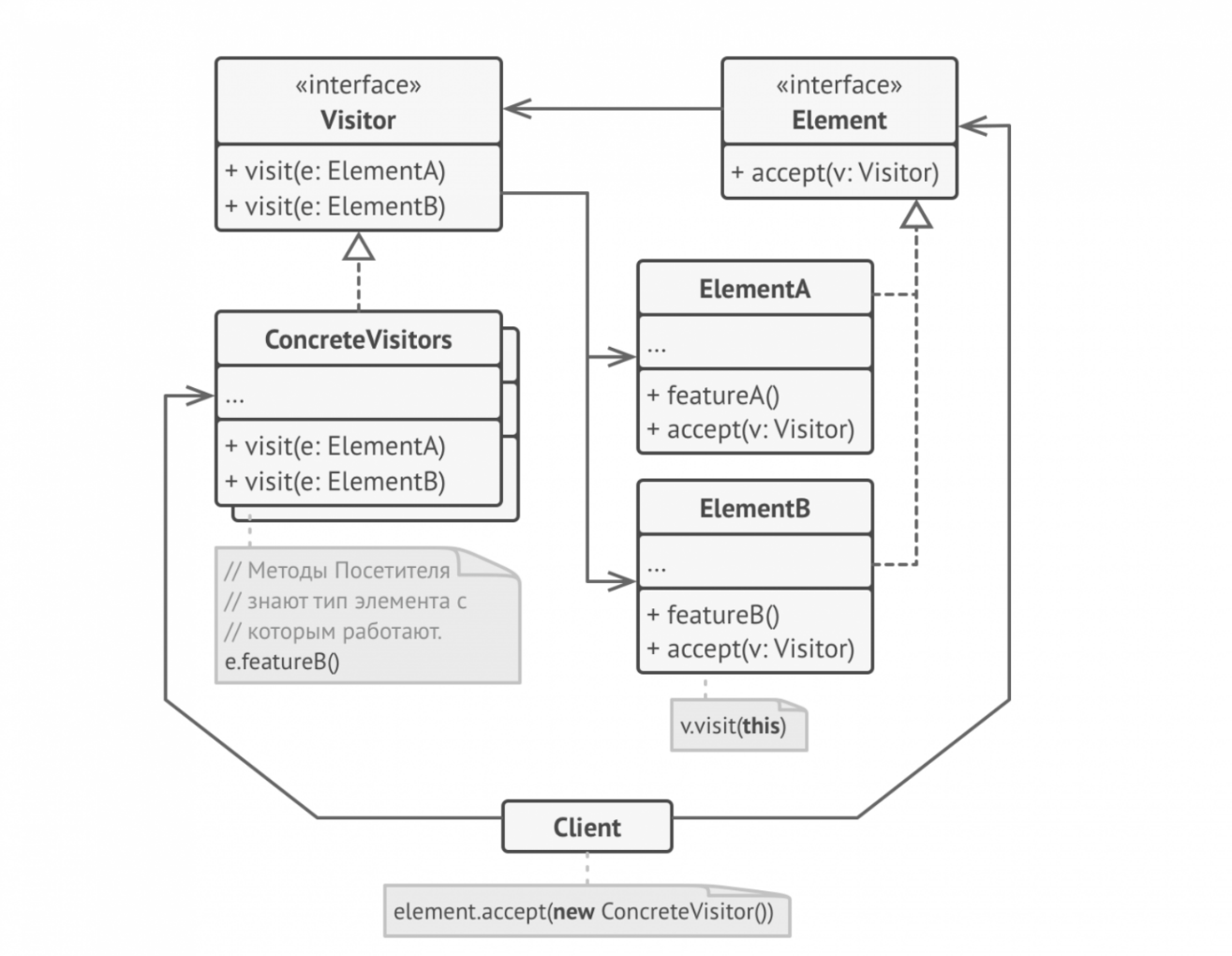


Диаграмма классов  
  
Самыми значимыми достоинствами этой диаграммы являются:

* экономия времени при объяснении задачи другим программистам;
* более точное и наглядное представление структуры основных элементов системы.

К минусам можно отнести значительные временные затраты при условии недостатка опыта работы с данной диаграммой.  
  
Подробнее о диаграмме классов можно прочитать [здесь](https://www.intuit.ru/studies/professional_skill_improvements/1364/courses/229/lecture/5954?page=4https://prog-cpp.ru/uml-classes/), а о паттерне «Посетитель» [здесь](https://refactoring.guru/ru/design-patterns/visitor).

Диаграмма деятельности

**Диаграмма деятельности** – это технология, позволяющая описывать логику процедур, бизнес-процессы и потоки работ. Во многих случаях они напоминают блок-схемы, но принципиальная разница между диаграммами деятельности и нотацией блок-схем заключается в том, что первые поддерживают параллельные процессы.

Если говорить кратко, то диаграмма деятельности помогает нам описать логику поведения системы. Можно построить несколько диаграмм деятельности для одной и той же системы, причем каждая из них будет фокусироваться на разных аспектах системы, показывать различные действия, выполняющиеся внутри нее.  
  
Именно на диаграмме деятельности представлены переходы от одной деятельности к другой. Это, по сути, разновидность диаграммы состояний, где все или большая часть состояний являются некоторыми активностями, а все или большая часть переходов срабатывают при завершении определенной деятельности и позволяют перейти к выполнению следующей. 

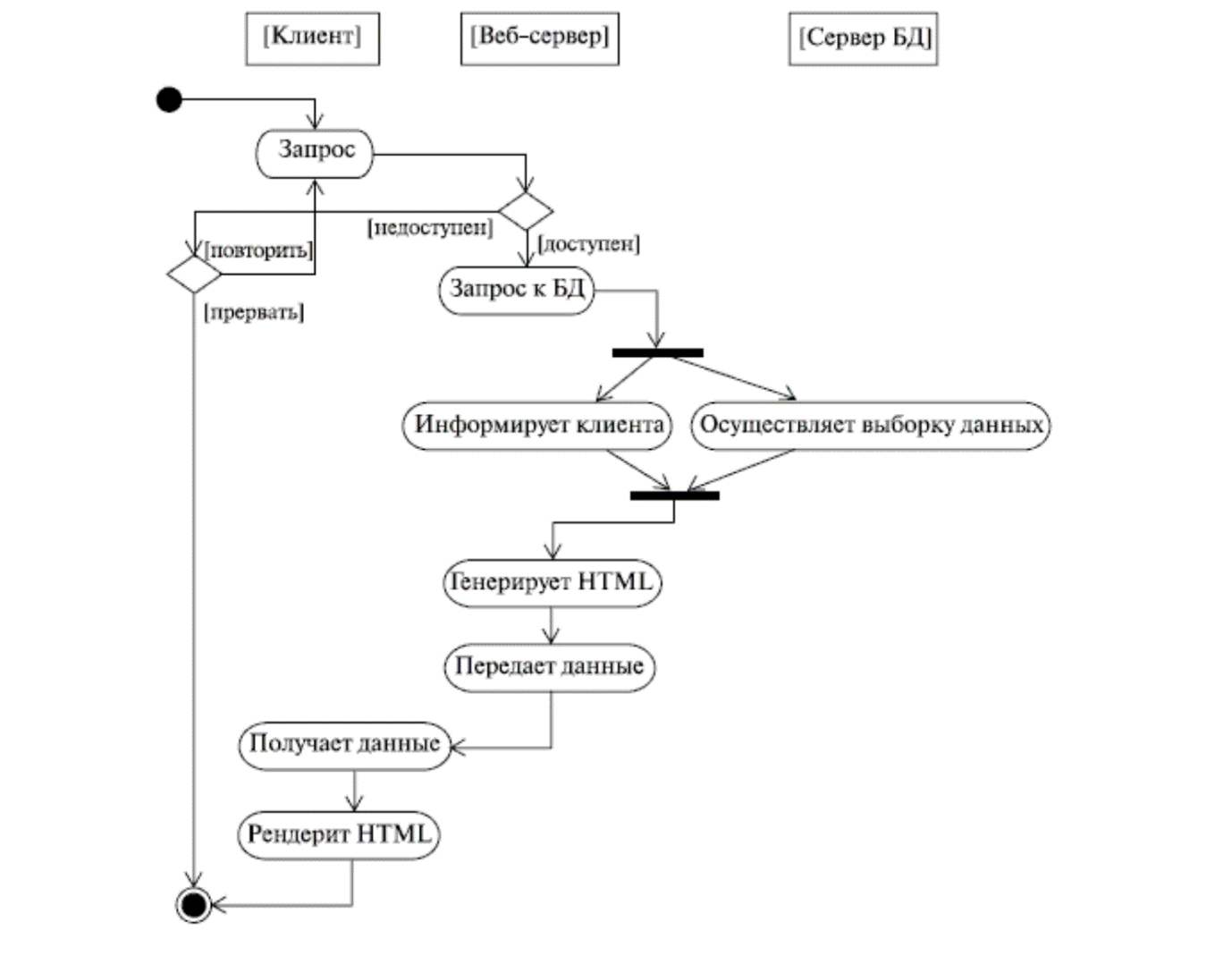


Диаграмма деятельности  
  
Смысл диаграммы вполне понятен. На ней показана работа с веб-приложением, которое решает некую задачу в удаленной базе данных. Обратите внимание на расположение активностей на этой диаграмме: они как бы разбросаны по трем колонкам, каждая из которых соответствует поведению одного из трех объектов — клиента, веб-сервера и сервера баз данных. Благодаря этому легко определить, каким из объектов выполняется каждая из деятельностей.  
  
Подробнее о диаграмме деятельности можно прочитать [здесь](https://litresp.ru/chitat/ru/%D0%9B/leonenkov-aleksandr/samouchitelj-uml/7).

Заключение

Надеюсь, что после этой статьи вы по-другому посмотрите на UML. Теперь при прочтении литературы или сайтов, посвященных данной теме, вам будет проще понять, какую цель преследует UML, и найти возможности для его применения. Попробуйте начать применять его и вы почувствуете всю силу и мощь, скрываемую за набором стрелок и квадратиков.