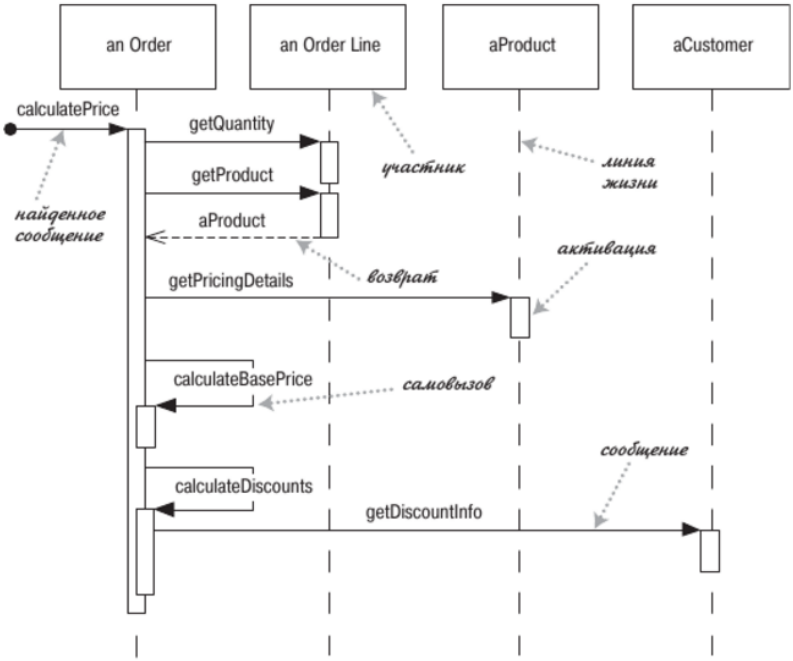
Практика 5: Моделирование поведения

Юрий Литвинов
yurii.litvinov@gmail.com

07.03.2022

1. Диаграммы последовательностей

На этом занятии предлагается порисовать несколько менее часто встречающиеся диаграммы, использующиеся в основном для моделирования поведения разрабатываемых систем. Первая из них — это диаграмма последовательностей (sequence diagram). Такие диаграммы очень полезны при анализе асинхронных и параллельных программ, вот напоминание их синтаксиса:



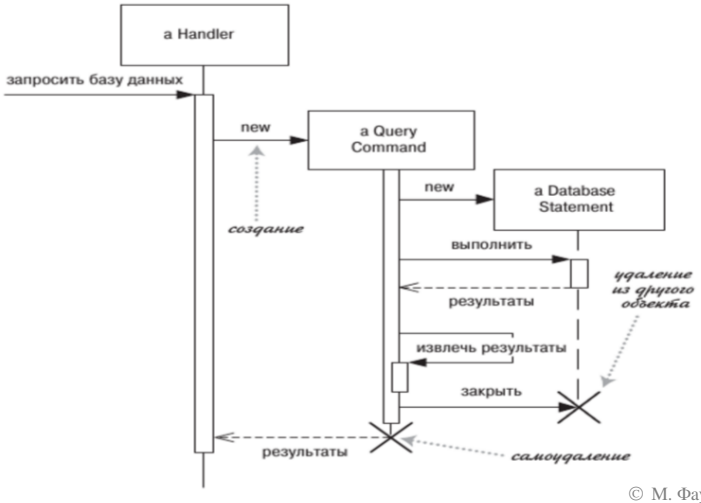
© М. Фаулер, UML. Основы

На что стоит обратить внимание при рисовании диаграмм на практике:

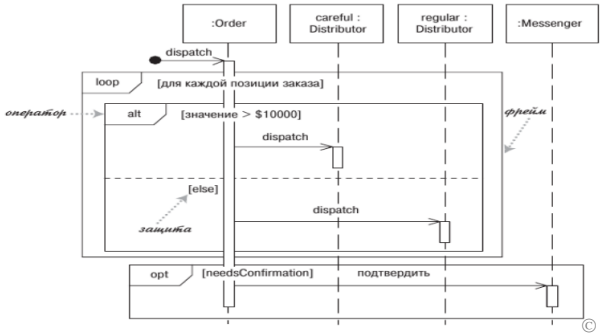
- сообщение может быть послано только активным объектом, поэтому не может выходить из просто линии жизни, нужно линию активации;

- сообщение надо обработать, поэтому его получение порождает линию активации хотя бы ненадолго;
- объект не может проснуться сам собой, так что линия активации может начаться только с приёма сообщения;
- если вы хотите смоделировать асинхронную систему, где непонятно, когда придёт то или иное сообщение, рисуйте его источник как актора — у него линия жизни вся покрыта активацией; или не рисуйте вовсе, есть «найденное сообщение», оно не требует источника;
- возврат имеет смысл рисовать, только если нам важно, что или когда вернули, в остальных случаях возврат просто подразумевается.

Можно показывать явное создание и удаление объектов:



А можно описать последовательность сообщений в зависимости от каких-то условий, циклы и даже полноценный алгоритм, с помощью фреймов:

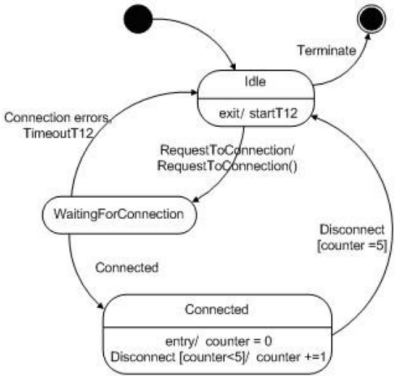


Вообще фреймы сильно ухудшают читабельность диаграммы, а для визуализации алгоритмов есть средства получше — диаграммы активностей, например. Так что фреймы — очень ситуационная штука.

2. Диаграммы конечных автоматов

Диаграммы конечных автоматов (также известные как диаграммы состояний) — это на самом деле несколько упрощённые диаграммы Харела, предложенные им ещё в 1987 году, которые попали в UML с минимальными изменениями. Они использовались и задолго до Харела, конечно, однако у Харела (и в UML) они хитрее, чем обычно ожидают от конечных автоматов, в частности, поддерживают вложенные и параллельные состояния.

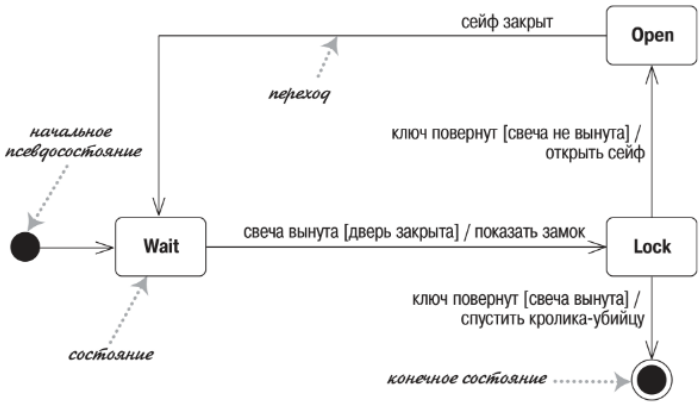
Напоминание о синтаксисе:



Внешне диаграммы конечных автоматов похожи на диаграммы активностей, поэтому их часто путают и из-за этого рисуют неправильно (в этом есть смысл, в UML до версии 2 эти диаграммы не разделялись). Есть важные семантические различия:

- на диаграмме активностей рисуются активности, система в них не задерживается, а сразу переходит дальше; на диаграмме конечных автоматов рисуются состояния — стабильные отрезки жизненного цикла объекта, в которых он находится большую часть времени и может из них выйти только если что-то произойдёт;
- полезная работа на диаграммах активностей производится в активностях, на диаграммах автоматов — как правило, при переходе;
- диаграммы активностей моделируют один метод объекта (или какую-то функцию или что-то такое), диаграммы конечных автоматов — целый объект (состояния моделируются полями объекта).

Более подробно про синтаксис:



© М. Фаулер, UML. Основы

Внутри состояния могут быть:

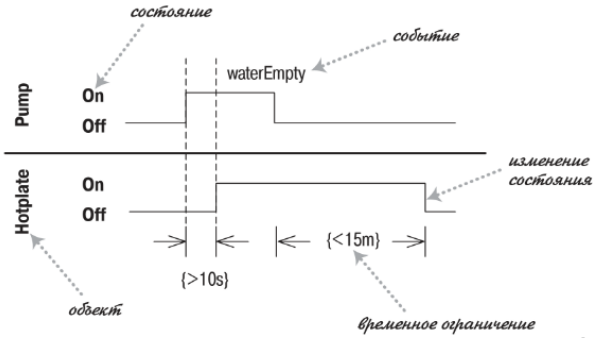
- entry activity — то, что делается при входе в состояние по любому из переходов;
- exit activity — то, что делается при выходе из состояния по любому исходящему переходу (и входная, и выходная деятельность — это, как правило, вызовы метода);
- do activity — деятельность, выполняющаяся всегда, когда система находится в таком-то состоянии (например, попытки подключения к сети для мобильного телефона);
- внутренний переход — переход по событию, который ведёт в то же состояние и не приводит к срабатыванию entry и exit activity. Переход вполне может быть полноценным переходом в то же состояние (рисуется как петля в графе), тогда entry и exit activity работают как обычно, хоть состояние и не меняется.

Событие, кстати, это нечто внешнее по отношению к системе, на что система может реагировать. Примеры событий — действие пользователя, сетевой пакет, считывание символа (если речь идёт об автоматном лексическом анализаторе, который, кстати, хоть и несколько необычный, но тоже пример реактивной системы, которая прекрасно моделируется конечными автоматами).

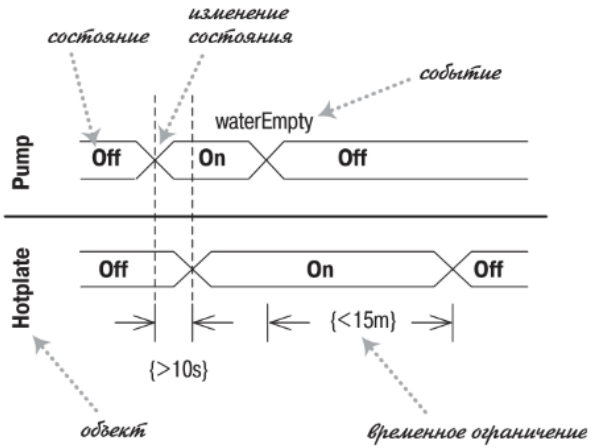
Надпись на переходе имеет следующий синтаксис: `[<trigger> [',' <trigger>]* ['[' <guard>']'] ['/' <behavior-expression>]]` — один переход может реагировать на несколько событий сразу, иметь опционального стражника (в квадратных скобках) и через слеш действие (вызов метода или отсылку к диаграмме активностей, которая поясняет, что нужно делать при переходе).

3. Временные диаграммы

Временные диаграммы создавались прежде всего для инженеров-электронщиков или для людей, проектирующих системы реального времени, в реальной практике не встречались автору ни разу. Есть два варианта синтаксиса:



и



В обоих случаях рисуется временная шкала (время идёт слева направо), на ней вертикально располагаются объекты, которые могут находиться в некоторых состояниях. Первый вариант показывает переключение состояния как скачок линии, второй — как пересечение линий с именем состояния внутри. Второй вариант удобнее, если состояний много, первый вариант нагляднее. Помимо состояний указываются и временные ограничения, в фигурных скобках, как принято в UML для ограничений.

Тонкость в том, что это один из немногих примеров *неграфовых* визуальных языков, поэтому многие редакторы их попросту не умеют. Умеет Visual Paradigm и вроде умеет Creately.

4. Задание на пару

Нарисовать следующие диаграммы:

1. диаграмму последовательностей регистрации и ремонта дефекта из уже знакомого вам запроса `https://bit.ly/defects-rfp`;

2. диаграмму конечных автоматов, описывающую поведение микроволновки;
3. временную диаграмму любого сценария работы микроволновки;
    - в VP это может быть не совсем тривиально: https://www.visual-paradigm.com/support/documents/vpuserguide/94/2586/6715_drawingtimin.html
    - в diagrams.net не факт, что вообще возможно, из браузерных решений, наверное https://creately.com/lp/timing-diagram-software/