Тема: UML – Диаграмма (видов) деятельности (Activity Diagram)

Содержание:

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Конечно, мы не всегда задумываемся о том, что подобные операции относятся к столь научным категориям. Например, чтобы позвонить по телефону, нам предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций, таких как разборка силового агрегата, снятие генератора и некоторых других.

Для моделирования процесса выполнения операций в языке UML используются так называемые **диаграммы деятельности**. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами — переходы от одного состояния действия к другому.

Именно они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий. Метамодель UML предоставляет для этого необходимые термины и семантику.

В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

1. Состояние действия

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и по крайней мере одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами (рис. 1). Внутри этой фигуры

записывается выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

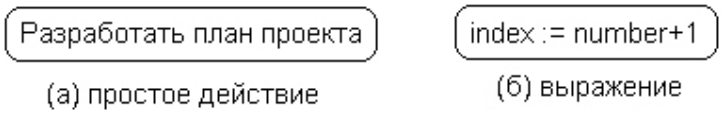


Рис. 1. Графическое изображение состояния действия

Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами (рис. 1, а). Если же действие может быть представлено в некотором формальном виде, то целесообразно записать его на том языке программирования, на котором предполагается реализовывать конкретный проект (рис. 1, б).

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния (см. рис. 2). При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии. Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное — в ее нижней части.

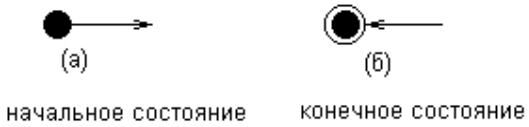


Рис. 2. Графическое изображение начального и конечного состояний

2. Переходы

При построении диаграммы деятельности используются переходы, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. Этот переход переводит деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, то сработать может только один из них. Именно в этом случае для каждого из таких переходов должно быть явно записано сторожевое условие в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата. Такая ситуация получила название ветвления, а для ее обозначения применяется специальный символ.

Графически ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста (рис. 3). В этот ромб может входить только одна стрелка от того состояния действия, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа ветвления. Выходящих стрелок может быть две или более, но для

каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

В качестве примера рассмотрим фрагмент известного алгоритма нахождения корней квадратного уравнения. В общем случае после приведения уравнения второй степени к каноническому виду: $a*x^2 + b*x + c = 0$ необходимо вычислить его дискриминант. Причем, в случае отрицательного дискриминанта уравнение не имеет решения на множестве действительных чисел, и дальнейшие вычисления должны быть прекращены. При неотрицательном дискриминанте уравнение имеет решение, корни которого могут быть получены на основе конкретной расчетной формулы.

Графически фрагмент процедуры вычисления корней квадратного уравнения может быть представлен в виде диаграммы деятельности с тремя состояниями действия и ветвлением (рис. 3). Каждый из переходов, выходящих из состояния "Вычислить дискриминант", имеет сторожевое условие, определяющее единственную ветвь, по которой может быть продолжен процесс вычисления корней в зависимости от знака дискриминанта. Очевидно, что в случае его отрицательности, мы сразу попадаем в конечное состояние, тем самым завершая выполнение алгоритма в целом.

Примечание

Строго говоря, первое из состояний рассматриваемого алгоритма следует считать состоянием под-деятельности, поскольку приведение квадратного уравнения к каноническому виду может потребовать нескольких элементарных действий (приведение подобных и перенос отдельных членов уравнения из правой его части в левую).

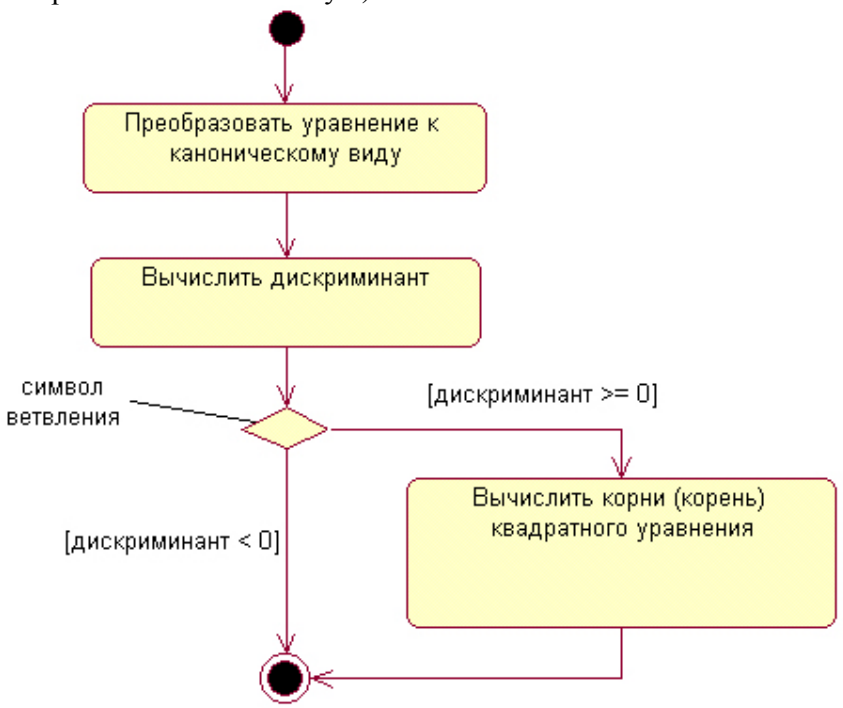


Рис. 3. Фрагмент диаграммы деятельности для алгоритма нахождения корней квадратного уравнения

В рассмотренном примере, как видно из рис. 3, выполняемые действия соединяются в конечном состоянии. Однако это вовсе не является обязательным. Можно изобразить еще один символ ветвления, который будет иметь несколько входящих переходов и один выходящий.

В следующем примере (рис. 4) рассчитывается общая стоимость товаров, покупаемых по кредитной карточке в супермаркете. Если эта стоимость превышает $50, то выполняется аутентификация личности владельца карточки. В случае положительной проверки (карточка действительная) или если стоимость товаров не превышает $50, происходит снятие суммы со счета и оплата стоимости товаров. При отрицательном результате (карточка недействительная) оплаты не происходит, и товар остается у продавца.

Примечание

Как видно из этого же рисунка, допускается использовать вместо сторожевого условия слово "иначе" (“else”), указывающее на тот переход, который должен сработать в случае невыполнения сторожевого условия ветвления.

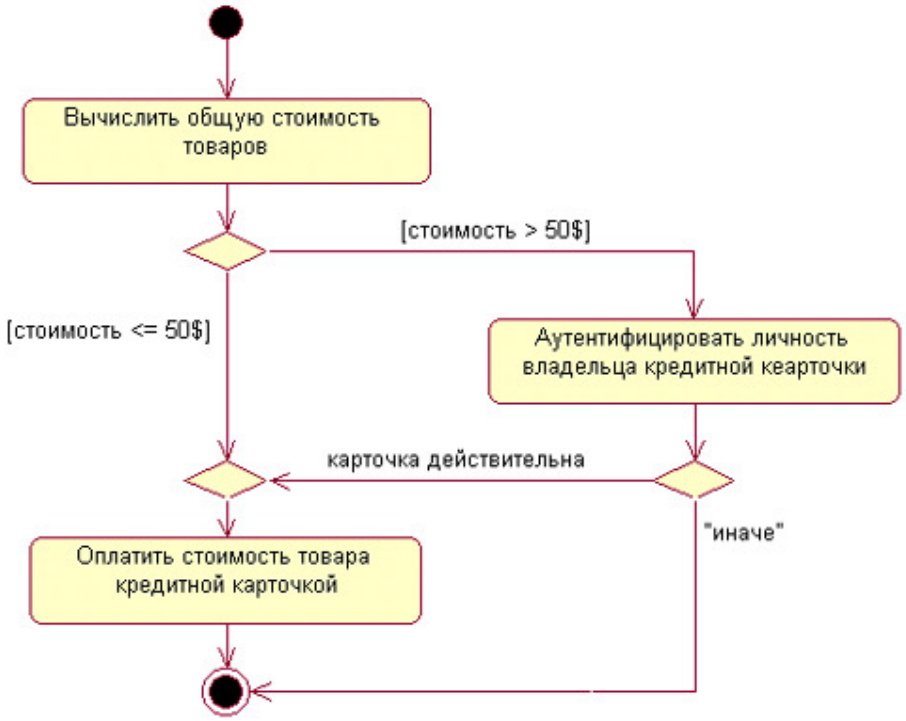


Рис. 4. Различные варианты ветвлений на диаграмме деятельности

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений. Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем, необходимы графические примитивы для представления параллельных процессов. В языке UML для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления. Таким символом является прямая черточка.

Как правило, такая черточка изображается отрезком горизонтальной линии, толщина которой несколько шире основных сплошных линий диаграммы деятельности. При этом разделение (concurrent fork) имеет один входящий переход и несколько выходящих (рис. 5, а). Слияние (concurrent join), наоборот, имеет несколько входящих переходов и один выходящий (рис. 5, б).

Для иллюстрации особенностей параллельных процессов выполнения действий рассмотрим ставший уже классическим пример с приготовлением напитка. Достоинство этого примера

состоит в том, что он практически не требует никаких дополнительных пояснений в силу своей очевидности (рис. 6).

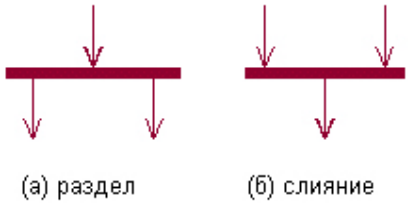


Рис. 5. Графическое изображение разделения и слияния параллельных потоков управления

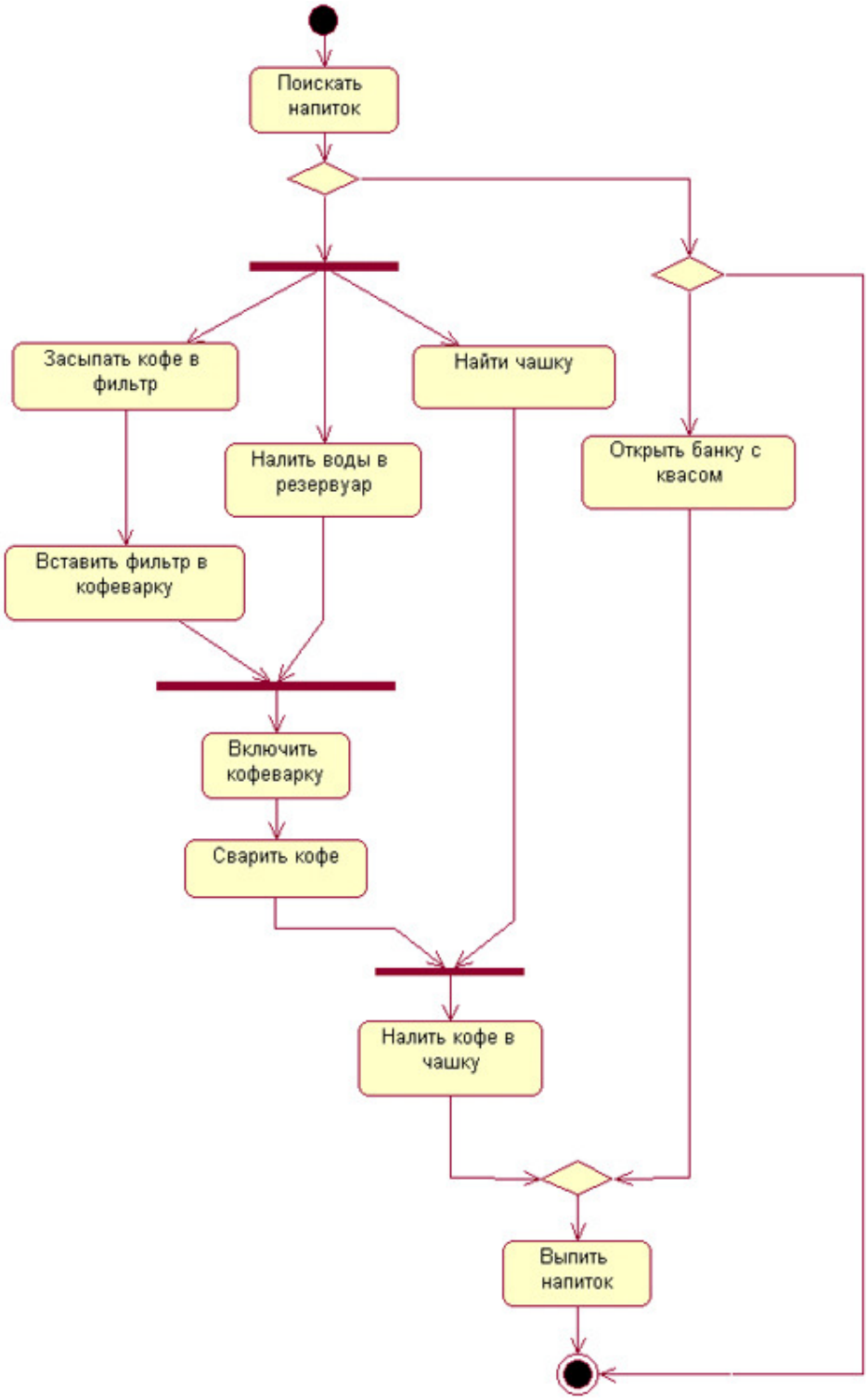


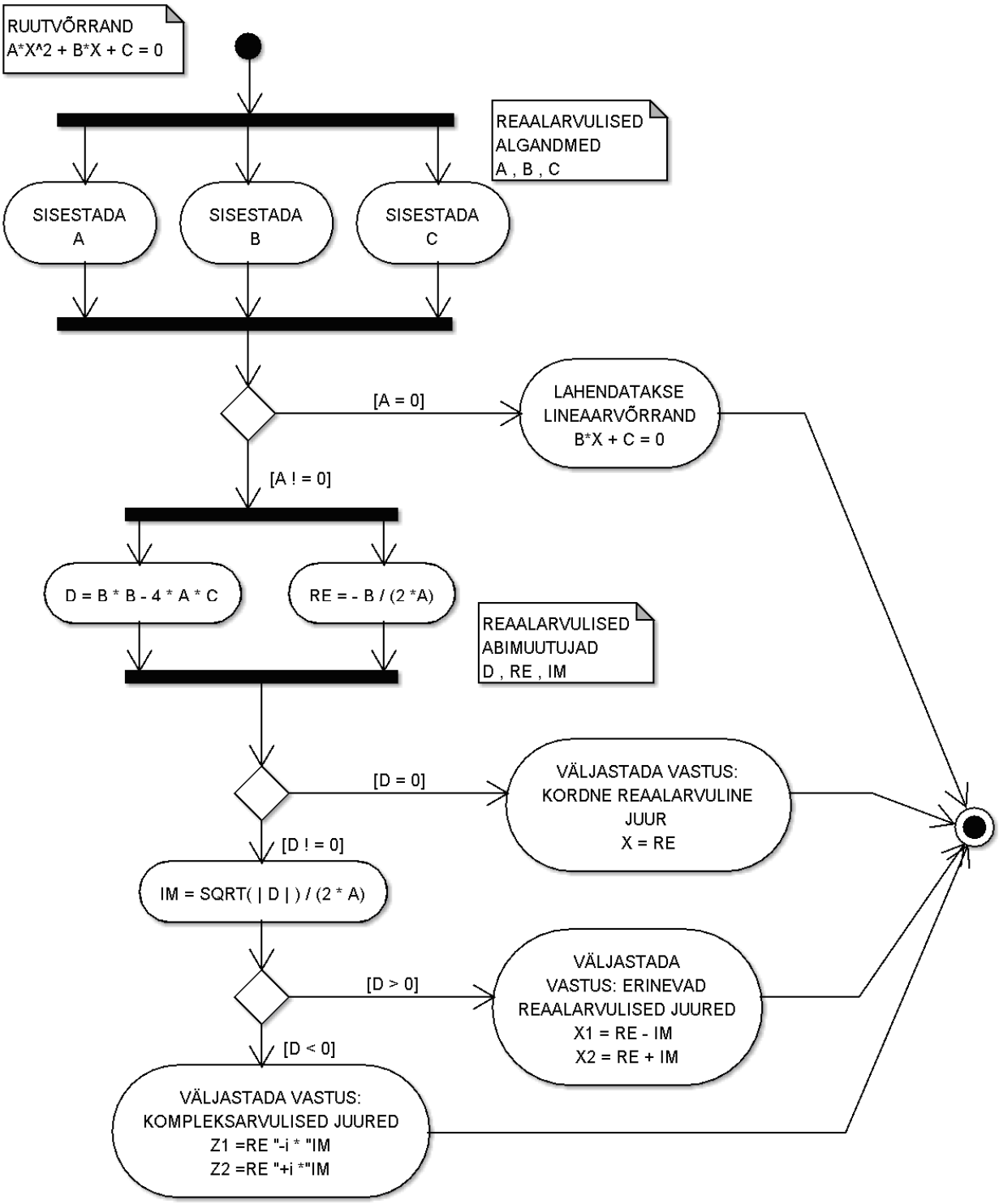
Рис. 6. Диаграмма деятельности для примера с приготовлением напитка

Примечание

Наличие параллельности проявляется в том, что мы можем заняться поисками чашки во время приготовления кофе.

NB! Представленная здесь информация по построению диаграмм видов деятельности (Activity Diagram) достаточна для решения задач курса «Programmeerimine 1». Дополнительную информацию о диаграммах UML см., например, на сайте http://www.uml.org

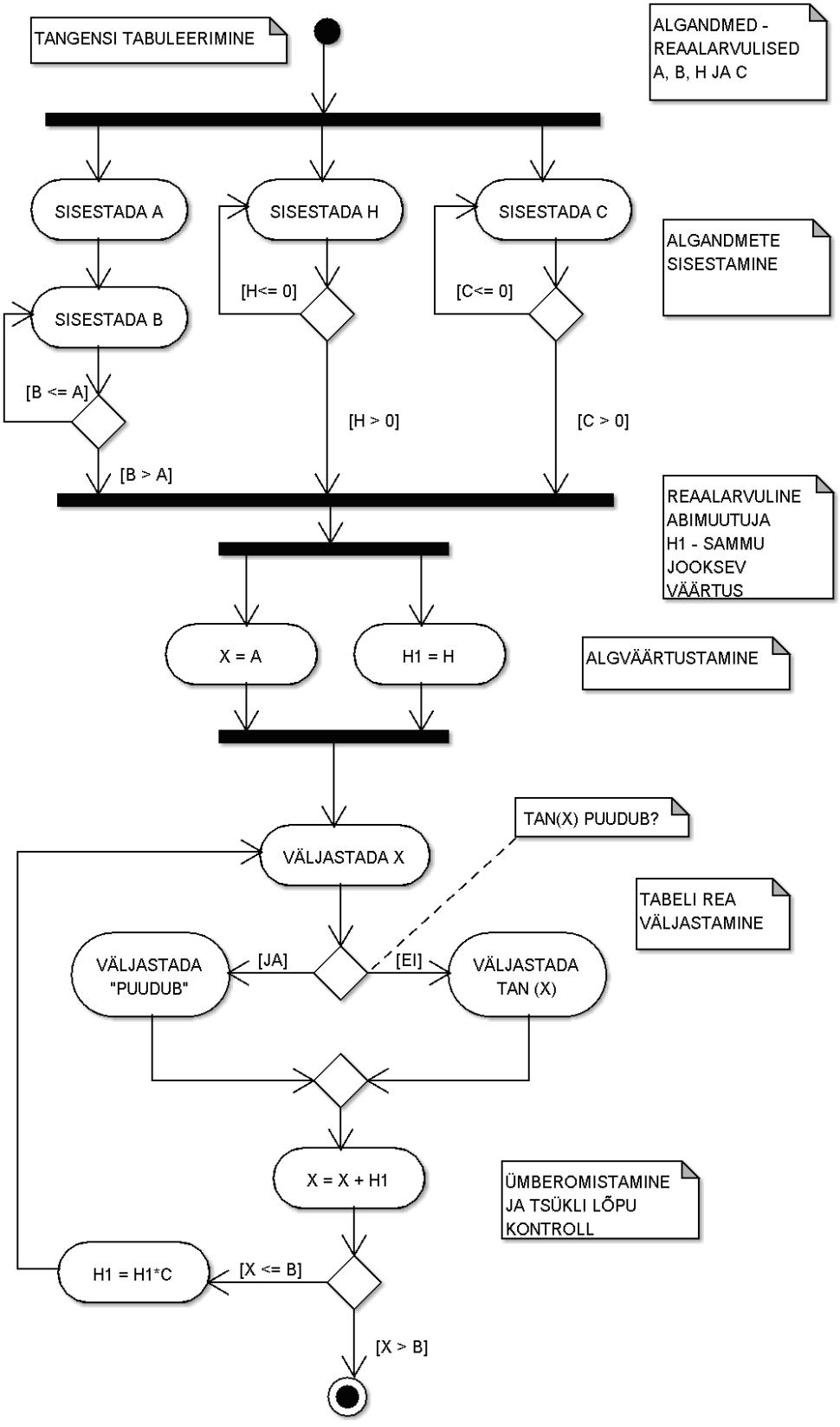
- **Упражнение 1***



На данном рисунке представлена диаграмма видов деятельности (**Activity Diagram**) решения квадратного уравнения. Предполагается, что, например, ввод данных, является параллельным

процессом. Перерисовать в ArgoUML диаграмму и дополнить ее частью, соответствующей решению линейного уравнения.

- **Упражнение 2** *



По **Activity Diagram** написать программу табулирования тангенса на языке **C**, используя операторы циклов.

Отчет оформляется в MS Word (или в другом текстовом редакторе), представляется в распечатанном виде. Отчет должен содержать необходимое для понимания описание работы. Представляемая программа должна быть прокомментирована.

Использованы материалы:

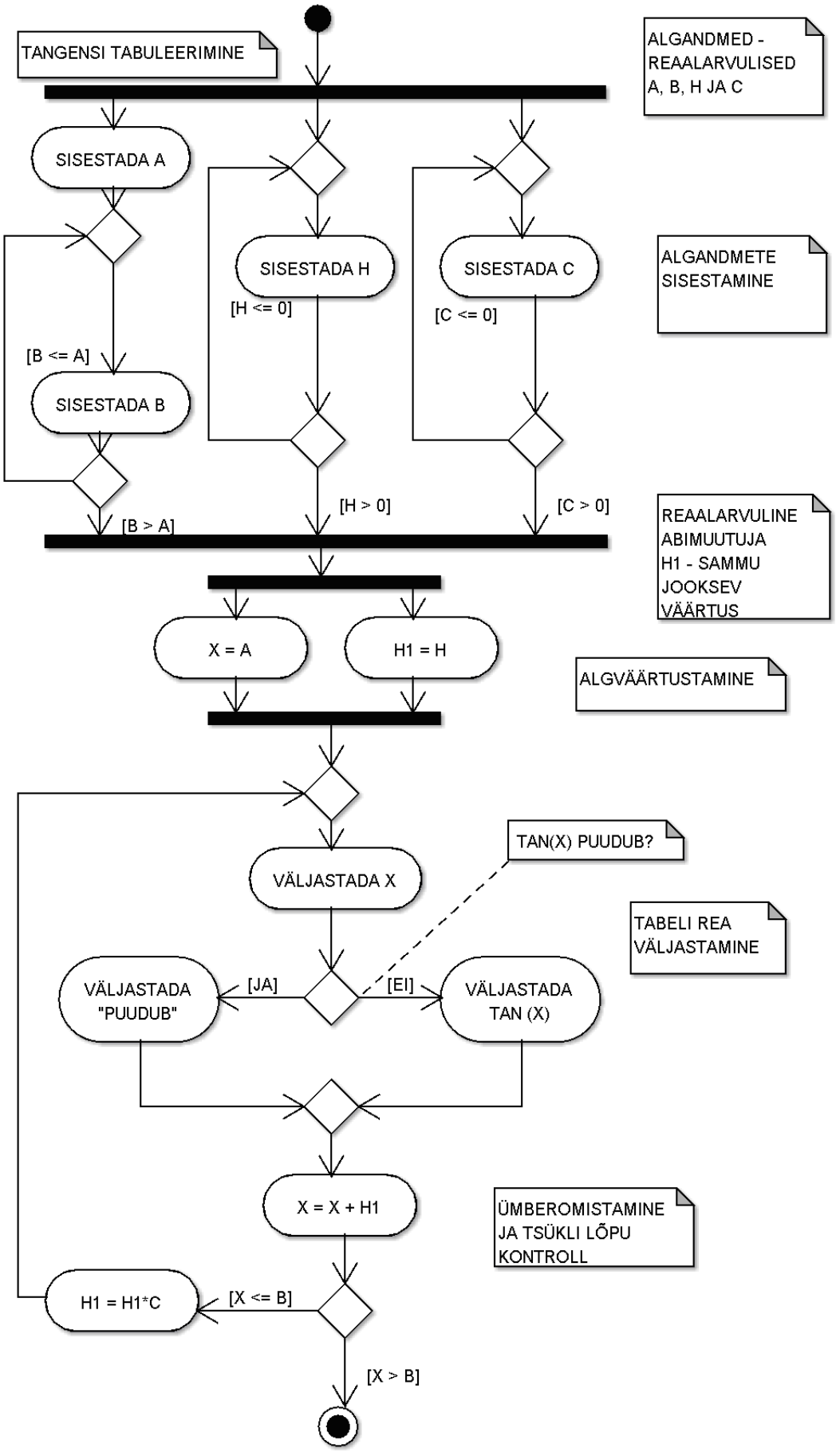
- *А. Леонков.* Самоучитель UML.
- Материалы *Ksenija Grigorjeva* (* рисунки в разделе «упражнения»; оба рисунка, находящиеся в данном документе, модифицированы – М.Б.)

Марина Брик

Составлено: 5.11.2007

Обновлено: 9.10.2008
18.10.2010

NB! По стандарту UML 2.0 (согласно, например, **Learning UML 2.0** By Kim Hamilton, Russell Miles) диаграмму упражнения 2 предпочтительнее перерисовать, используя вершины слияния (диаграмма упражнения 2 использует в качестве примера *только* одну такую вершину в цикле вывода строк таблицы):


TANGENSI TABULEERIMINE
ALGANDMED - REAALARVULISED A, B, H JA C
SISESTADA A
SISESTADA H
SISESTADA C
ALGANDMETE SISESTAMINE
[H <= 0]
[C <= 0]
[B <= A]
SISESTADA B
[B > A]
[H > 0]
[C > 0]
REAALARVULINE ABIMUUTUJA H1 - SAMMU JOOKSEV VÄÄRTUS
X = A
H1 = H
ALGVÄÄRTUSTAMINE
TAN(X) PUUDUB?
VÄLJASTADA X
TABELI REA VÄLJASTAMINE
VÄLJASTADA "PUUDUB"
[JA]
[EI]
VÄLJASTADA TAN (X)
X = X + H1
ÜMBEROMISTAMINE JA TSÜKLI LÕPU KONTROLL
H1 = H1*C
[X <= B]
[X > B]