Руководство по программе «ВизуальнаяАйвика»

Сорокин Давид Эрнестович <davsor@mail.ru>,
Россия, Республика Марий Эл, Йошкар-Ола

22 августа 2026 г.

Оглавление

1 Введение **7**

2 Как создать простую модель **15**

3 Как создать потоковую модель SFM **23**

4 Как создать модель системы массового обслуживания **31**

5 Язык моделирования **41**
5.1 Параметры моделирования 41
5.2 Переменные 43
5.3 Уравнения 44
5.4 Операторы 45
5.5 Постоянные 46
5.6 Функции 46
5.7 Диапазоны 56
5.8 Массивы 56
5.8.1 Функции для массивов и диапазонов 58
5.8.2 Инициализация массива 59
5.8.3 Отображение массивов на графиках 59
5.9 Анализ чувствительности 60
5.10 Транзакты 63
5.11 Поток внешних событий 63
5.12 Блок генератора 65
5.13 Блоки 66
5.14 Запуск блоков 69
5.15 Очереди 71
5.16 Прибор 74

5.17 Многоканальное устройство 78
5.18 Ансамбль транзактов 82
5.19 Связующая цепочка 83
5.20 Сигналы 88
5.21 Статистика 90
5.21.1 Статистика на основе наблюдений 90
5.21.2 Статистика с привязкой ко времени 92
5.22 Изменяемая ссылка 94
5.22.1 Эффективная проверка условных выражений . . 95
5.22.2 Особенности модельного времени 96
5.23 Функции конвейера 97
5.24 Счетчик транзактов 99
5.25 Строки 100
5.26 Вложенные модули 101

6 Вывод результатов 103
6.1 График отклонения 103
6.2 График экстремумов 105
6.3 Временной ряд 107
6.4 График XY 108
6.5 Таблица CSV 109
6.6 Последние значения 110
6.7 Гистограмма последних значений 110
6.8 Таблица CSV последних значений 111
6.9 Сводная статистика по последним значениям 112

7 Потоки и накопители 115
7.1 Накопитель интеграла 115
7.2 Массив интегралов 117
7.3 Массив накопителей интеграла 118
7.4 Накопитель конвейера 120
7.5 Массив накопителей конвейера 123

8 Частичное моделирование 127

9 Гибридное моделирование 131

10 Экспорт приложений .NET 137

11 Внешние модули **139**

A Детали реализации прибора **141**
A.1 Вытеснение прибора 141
A.2 Захват прибора 143
A.3 Освобождение прибора 143
A.4 Возврат прибора 143

Глава 1

Введение

«ВизуальнаяАйвика» (английское название «VisualAivika») — это программное средство визуального моделирования. Оно ориентировано на системную динамику и на дискретно-событийное моделирование. Есть простой в использовании редактор диаграмм, который позволяет создавать приятные на вид диаграммы потоков и накопителей (*англ.* Stock and Flow Maps, SFM), как показано на рисунке 1.1. Такие диаграммы также могут быть использованы для дискретно-событийных моделей.

Также существует моделирующий компонент, который поддерживает свой собственный высокоуровневый язык моделирования. Уравнения модели отображаются в отдельных вкладках, как демонстрирует рисунок 1.2. Пользователь может переключаться между диаграммами и уравнениями.

Есть редактор уравнений, где вы можете задавать интегралы, массивы, случайные функции — взгляните на рисунок 1.3. Редактор открывается сразу после выбора одного из элементов на диаграмме.

Более того, «ВизуальнаяАйвика» поддерживает вычислительный эксперимент по методу Монте-Карло, что позволяет проводить анализ чувствительности модели. Существуют средства для вывода графиков на диаграммы, чтобы показать результаты моделирования в виде временных рядов (*англ.* Time Series и XY Chart) и графика отклонения для тренда с доверительным интервалом (*англ.* Deviation Chart), как показано на рисунке 1.4. Результаты могут быть экспортированы как файлы данных в формате CSV.

Помимо этого, «ВизуальнаяАйвика» поддерживает массивы и ин-

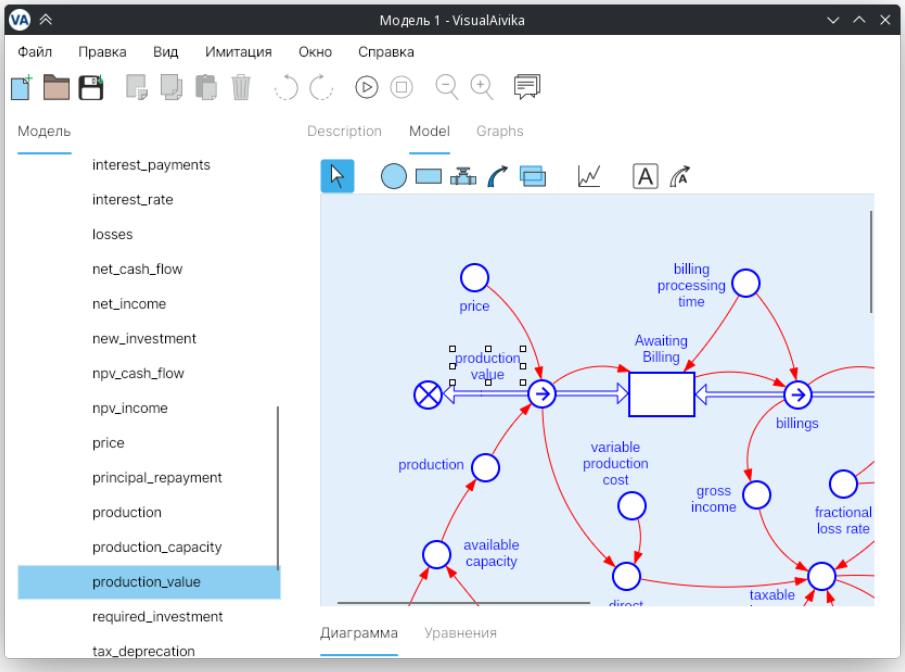


Рис. 1.1: Редактирование потоковой диаграммы SFM.

дексы. Рисунок 1.5 показывает график, который соответствует некоторому массиву.

Наконец, перед началом имитации пользователь можете задать параметры моделирования, как изображено на рисунке 1.6.

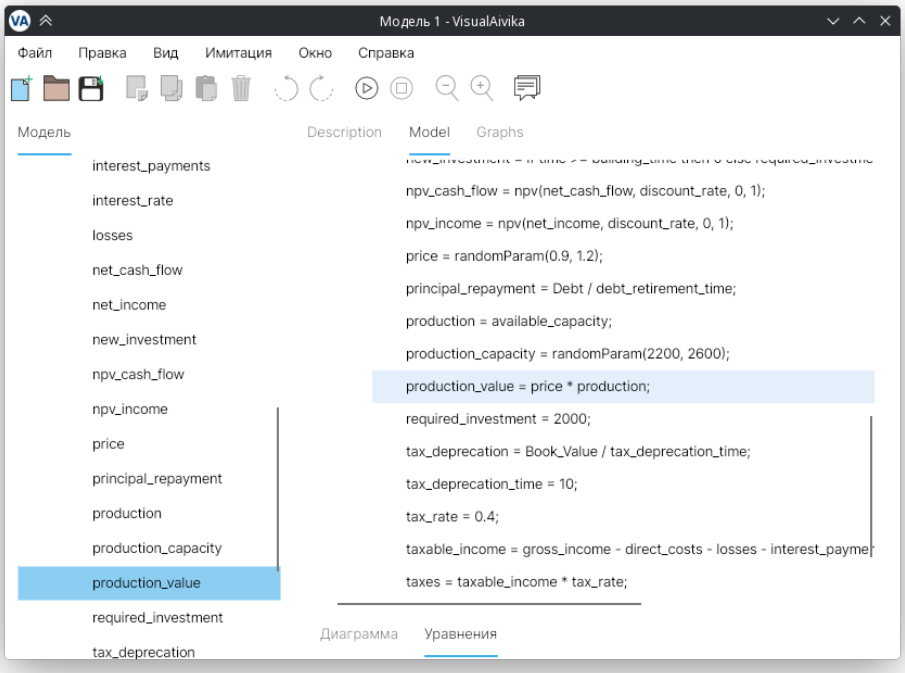


Рис. 1.2: Панель уравнений.

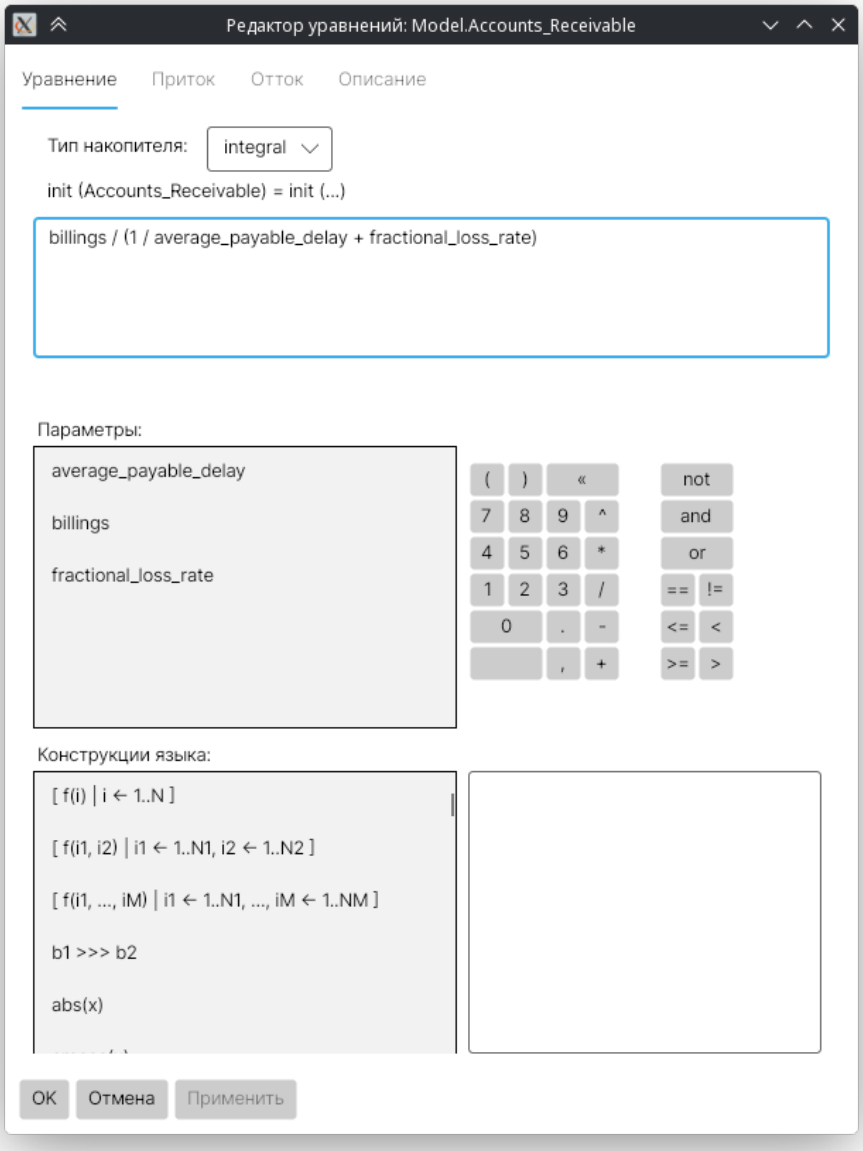


Рис. 1.3: Панель редактора уравнений.

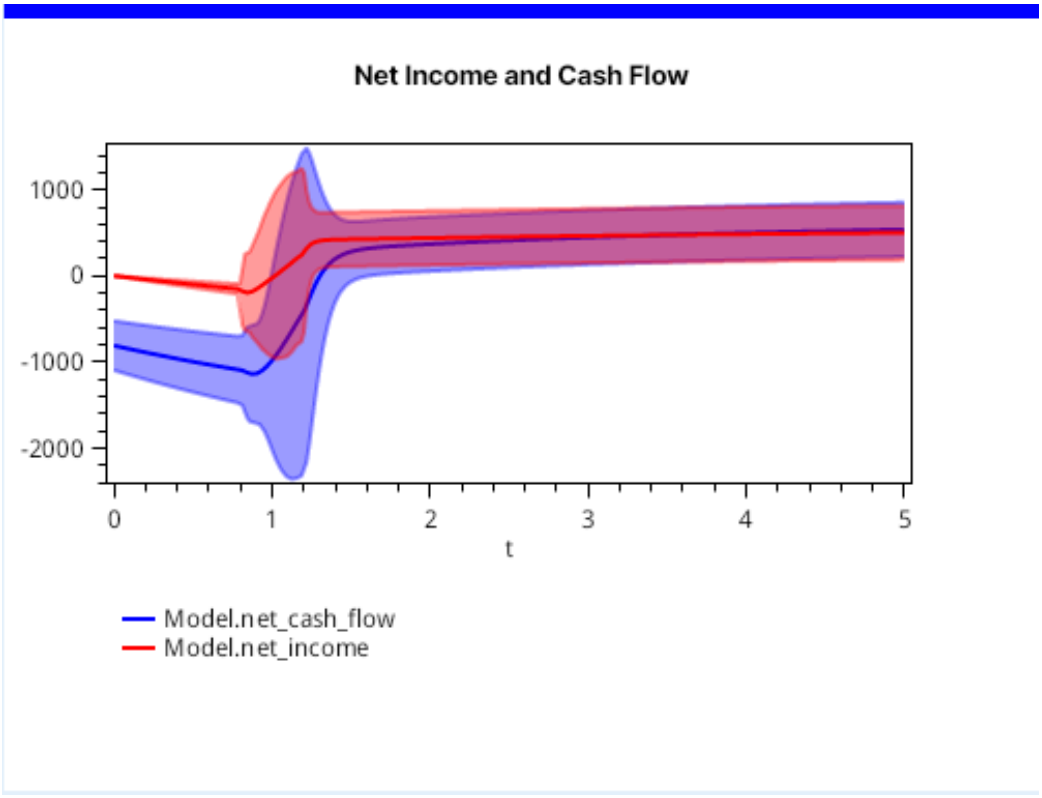


Рис. 1.4: График трендов с доверительными интервалами для анализа чувствительности.

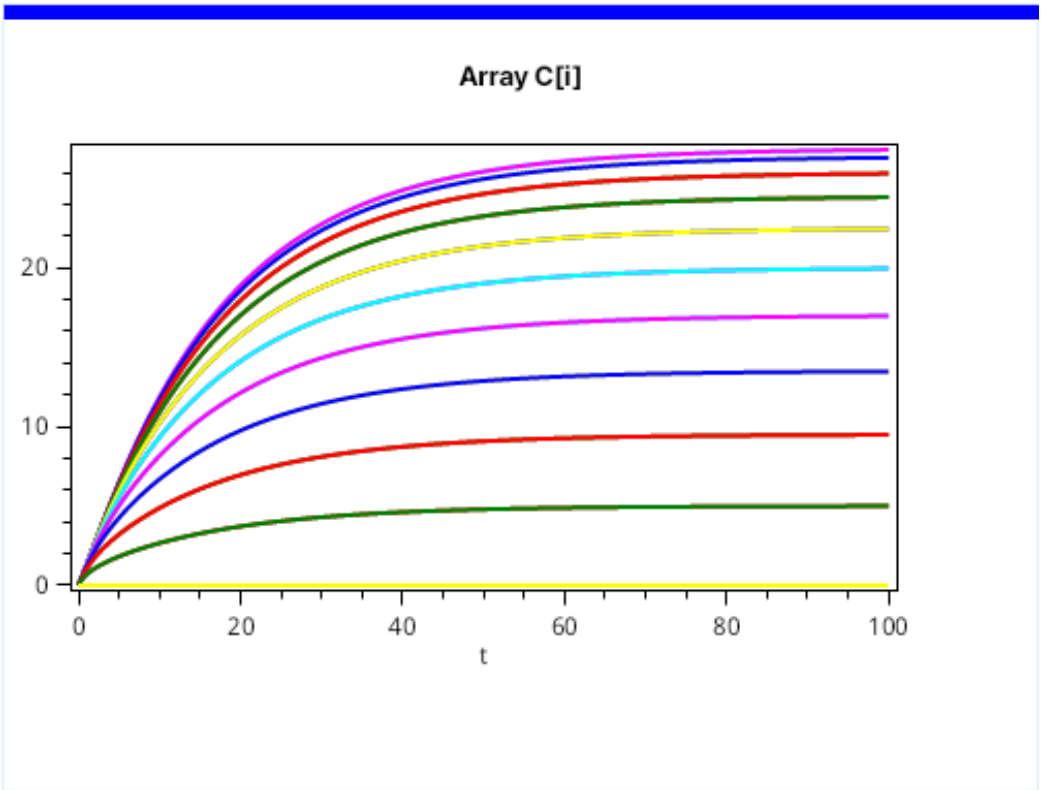


Рис. 1.5: Отображение массивов на графике.

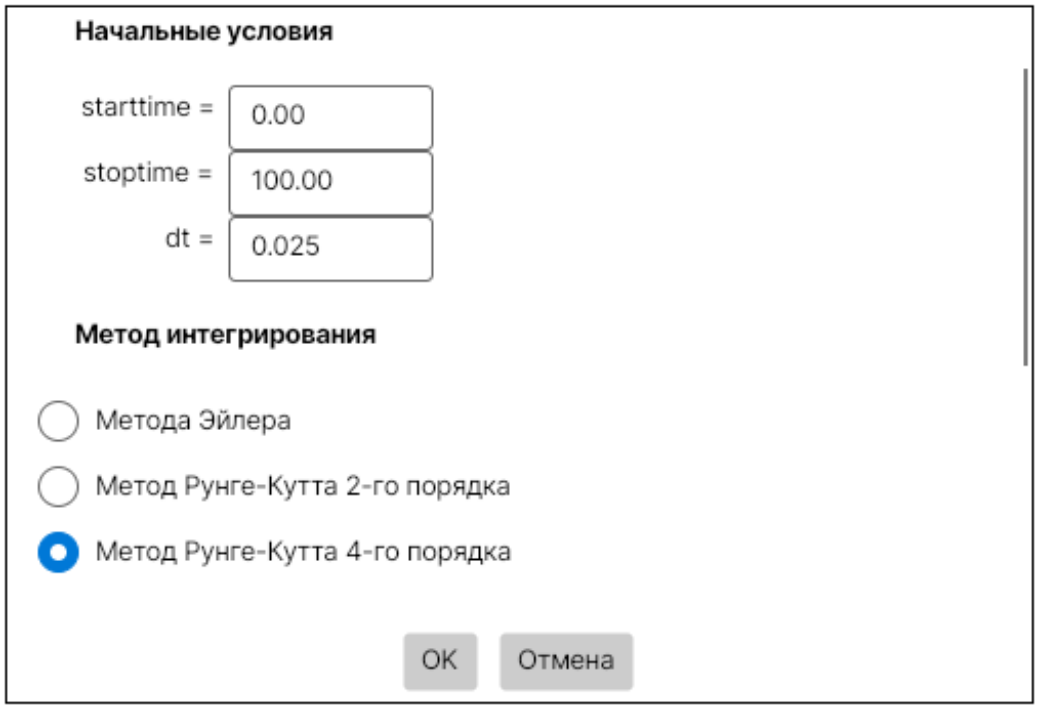


Рис. 1.6: Определение параметров моделирования.

Глава 2

Как создать простую модель

Например, мы можем воспроизвести следующие уравнения из модели «5-Minute Tutorial» для программы Berkeley-Madonna[1].

$$\begin{aligned} \dot{a} &= -ka \times a, \qquad a(t_0) = 100, \\ \dot{b} &= ka \times a - kb \times b, \qquad b(t_0) = 0, \\ \dot{c} &= kb \times b, \qquad c(t_0) = 0, \\ ka &= 1, \\ kb &= 1. \end{aligned}$$

Есть два способа, как определить эти уравнения. Первый способ основан на непосредственном использовании интегралов.

Создайте новую пустую модель и определите следующие дополнительные элементы SFM (*англ.* SFM Auxiliaries) и элементы соединений SFM (*англ.* SFM Links), как показано на рисунке 2.1. Чтобы добавить новые элементы, вы можете использовать панель инструментов диаграммы.

Затем щелкните мышкой по вкладке *Уравнения*. Далее щелкая по уравнениям переменных, завершите определение соответствующих уравнений в редакторе уравнений. В первый раз необходимо щелкнуть мышкой по одному из уравнений. Затем редактор останется открытым, и тогда достаточно будет щелкать мышкой по каждому следующему уравнению переменной. В итоге вы должны получить следующие уравнения, как показано на рисунке 2.2.

Сейчас настало время, чтобы добавить график. Вернитесь к панели диаграммы, щелкнув по вкладке *Диаграмма*. Выберите на панели инструментов диаграммы *Элемент для вывода результатов* и добавьте такой элемент на ту же диаграмму. Здесь вы должны получить нечто похожее на то, что изображено на рисунке 2.3. Выделенный фрагмент на изображенной диаграмме — это будущий элемент графика, который мы далее определим.

Затем щелкните мышкой по будущему элементу графика, чтобы открыть редактор графика. В этом редакторе мы можем добавить следующие переменные на график: A, B и C. Это должно выглядеть как на рисунке 2.4. Эти переменные являются интегралами.

Сейчас мы готовы для запуска имитации. На главной панели инструментов есть кнопка *Запуска имитации*, которая выглядит как стрелка, которая выделена красным цветом на рисунке 2.5. Нажмите на кнопку, чтобы запустить имитацию!

Вы должны увидеть результаты похожие на рисунок 2.6.

Возможно, что кривые на вашем графике выглядят не настолько гладко, как на рисунке. Чтобы это исправить и улучшить точность графиков, откройте пункт меню *Имитация / Параметры моделирования* и уменьшите значение параметра моделирования `dt`. Повторите запуск имитации. Сейчас вы должны получить более гладкий и точный график, как было показано на рисунке ранее.

Существует также другой метод задания обыкновенных дифференциальных уравнений, основанный на использовании диаграммы SFM потоков и накопителей (потоковой диаграммы).

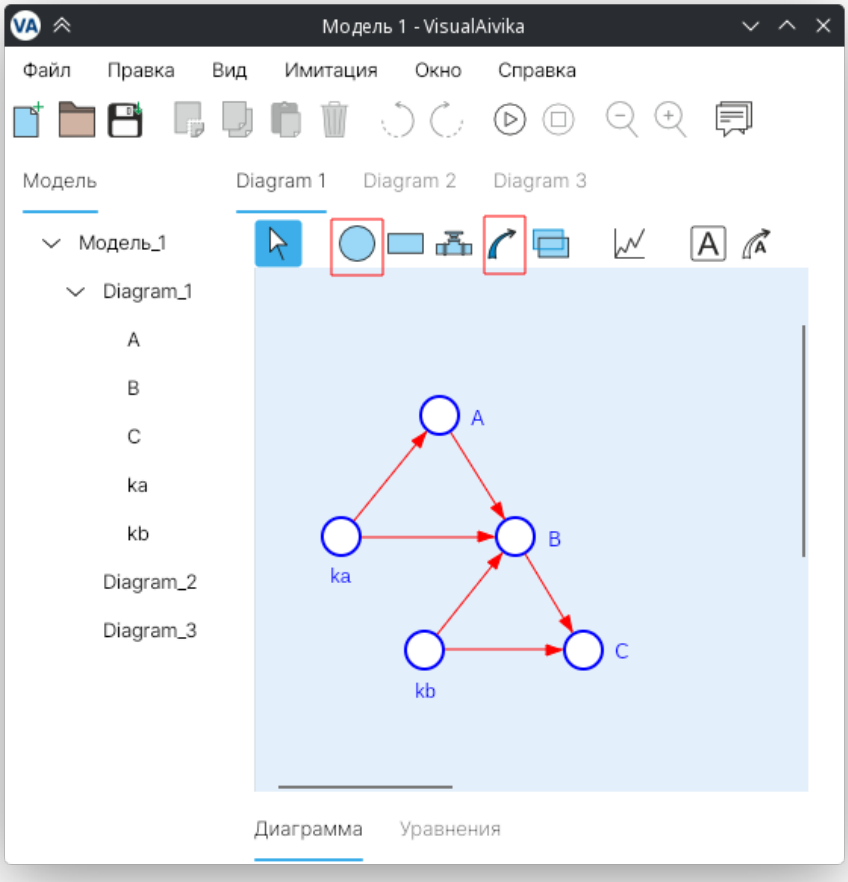


Рис. 2.1: Диаграмма состоит из будущих интегралов.

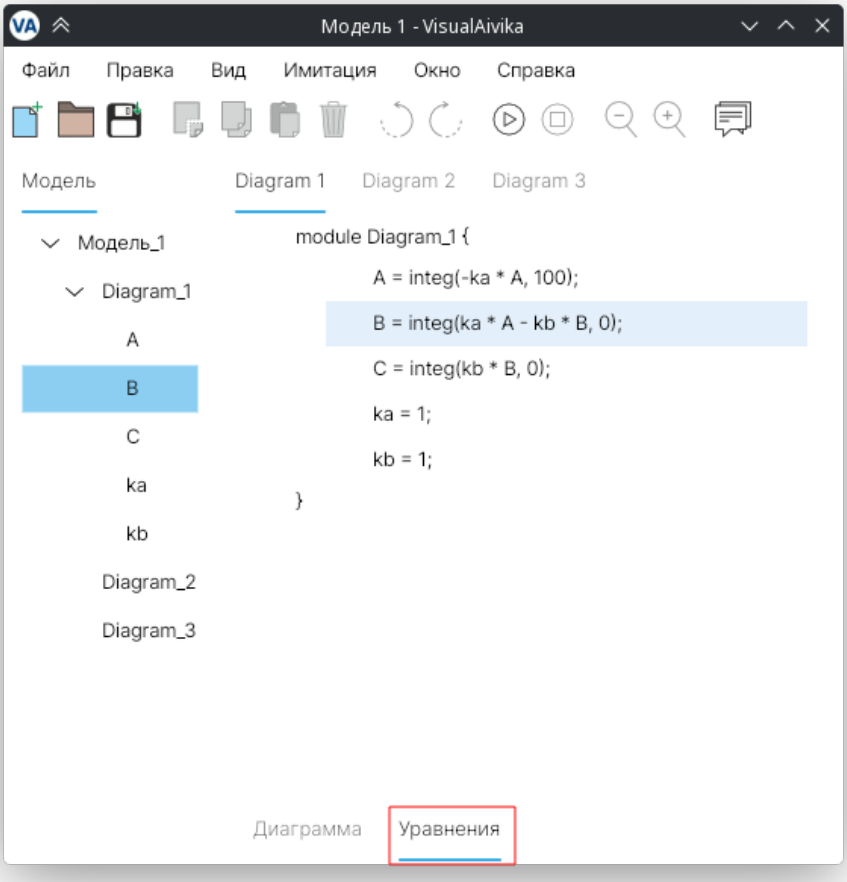


Рис. 2.2: Уравнения, состоящие из интегралов.

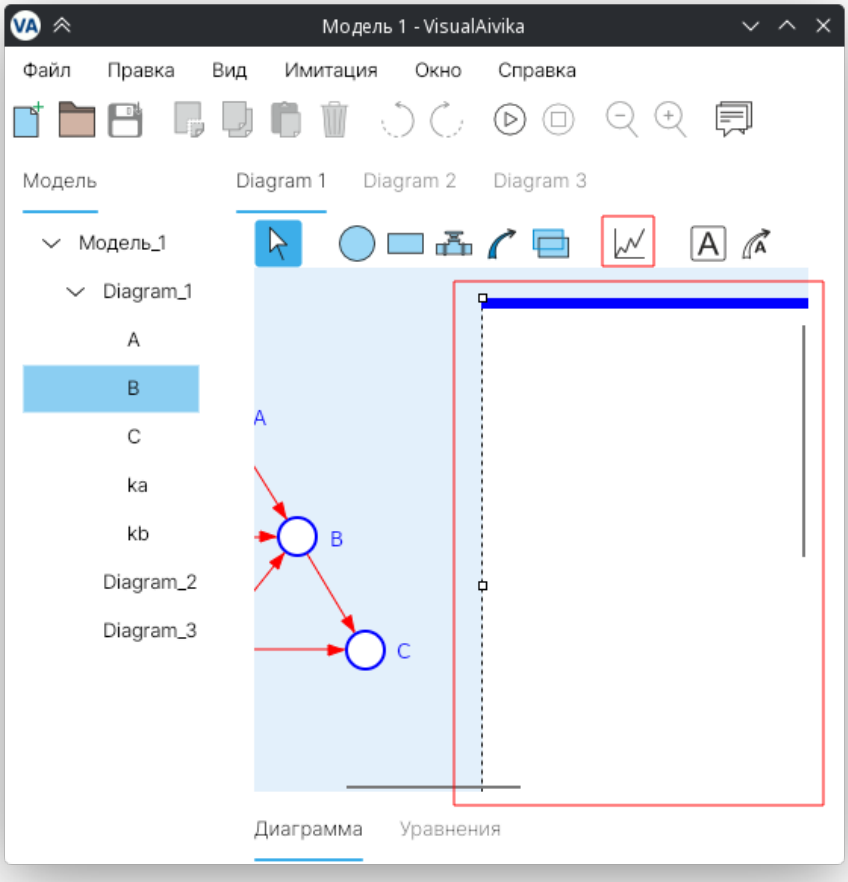


Рис. 2.3: После того, как график добавлен на диаграмму.

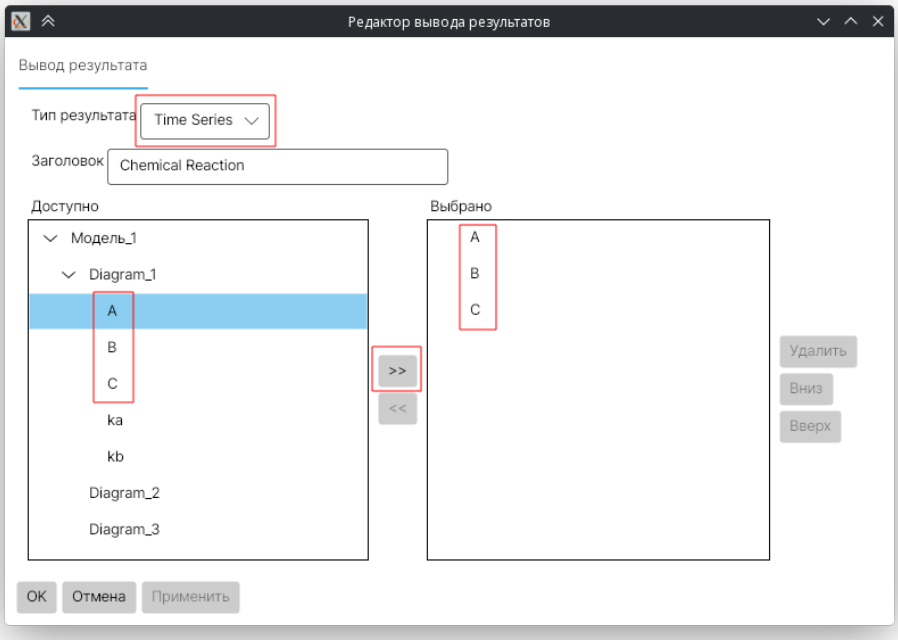


Рис. 2.4: Добавьте переменные интегралов на график для отображения.

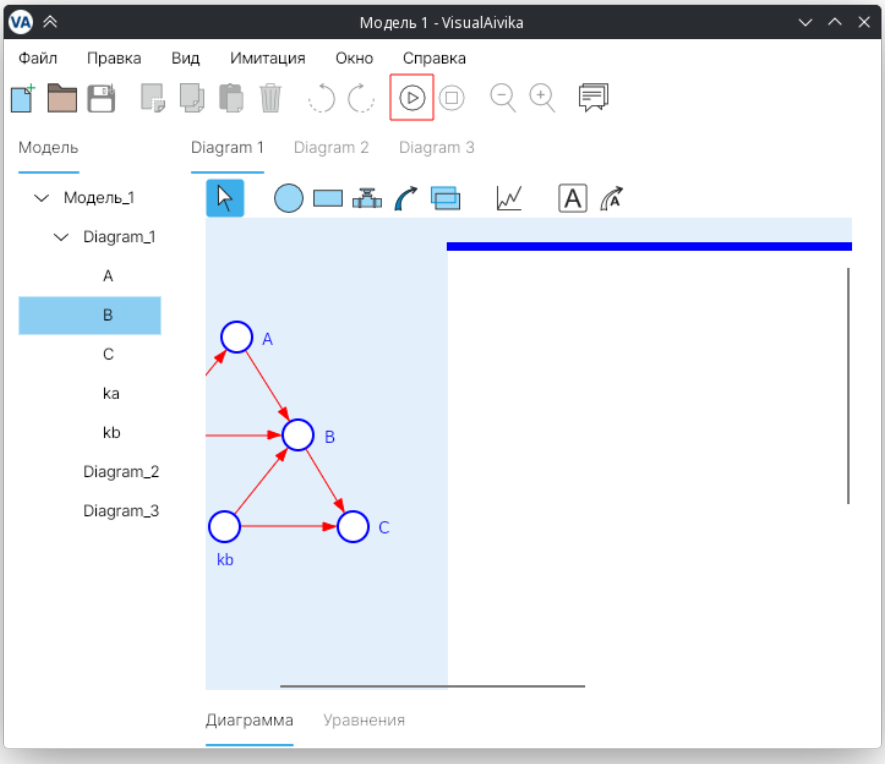


Рис. 2.5: Кнопка *Запуска имитации*.

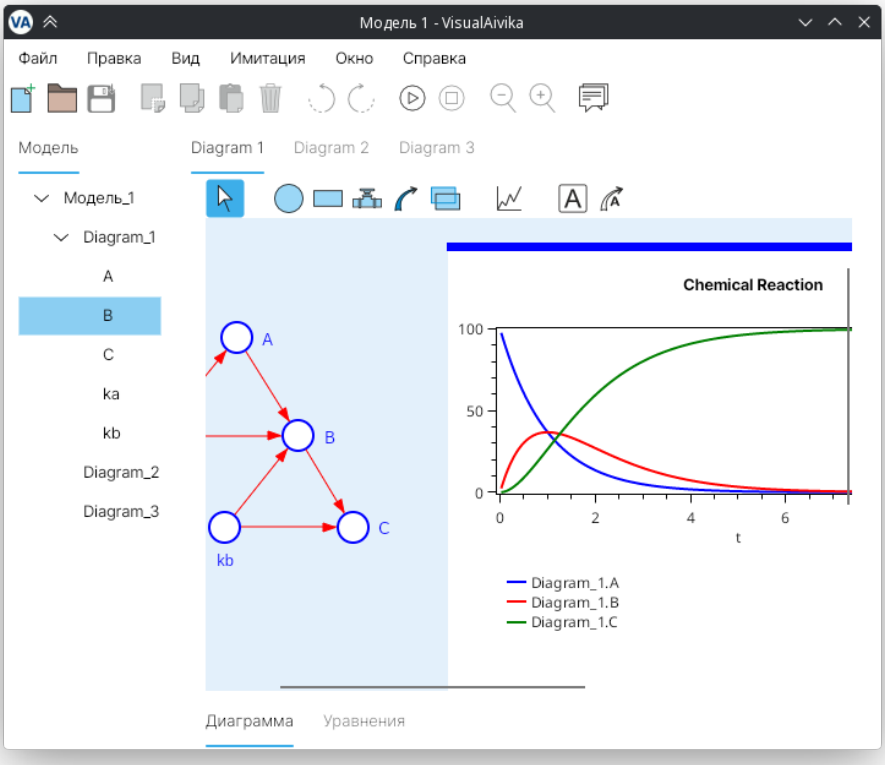


Рис. 2.6: Диаграмма после завершения имитации.

Глава 3

Как создать потоковую модель SFM

Накопитель SFM может быть резервуаром. Тогда он становится эквивалентным интегралу, а потоки SFM становятся эквивалентными производным. Направление потока SFM в таком случае показывает знак производной, который может быть положительным или отрицательным. Сочетание всех потоков SFM, подсоединенных к заданному накопителю SFM, задает соответствующую сумму производных с возможными разными знаками.

Однонаправленный поток SFM всегда несет в себе неотрицательное значение, тогда как двунаправленный поток может иметь значение любого знака. В то же время, реальный вклад на производную для каждого подсоединенного интеграла зависит от сочетания всех факторов: является ли поток SFM двунаправленным или однонаправленным, является ли поток входящим или исходящим.

К счастью, вкладка *Уравнения* отображает соответствующие уравнения математически ориентированным способом, где достаточно легко увидеть знак каждой производной и ее реальный вклад.

Для демонстрации подхода мы возьмем простую модель экспоненциального роста.

Создайте новую модель и, используя панель инструментов диаграммы, добавьте следующие элементы SFM на диаграмму, как показано на рисунке 3.1.

Щелкните мышкой по накопителю Cash, чтобы открыть *Редактор уравнений*. Определите начальное значение равное 1000, как иллюстри-

рует рисунок 3.2. Это значение будет начальным значением интеграла, который соответствует накопителю Cash.

Затем выберите поток Net Interest и изменить значение свойства *Направление потока* в редакторе диаграмм так, чтобы значение свойства было равным *uniflow*. Вы должны увидеть что-то похожее на рисунок 3.3.

Это означает, что соответствующий поток SFM может иметь только неотрицательное значение, хотя реальное влияние на накопитель SFM может отличаться по знаку, что зависит также от направления потока SFM. Здесь сущность Net Interest является входящим потоком для накопителя Cash. Следовательно, соответствующая производная имеет неотрицательный знак. Значение накопителя должно расти, как мы увидим далее.

После того, как мы превратили поток Net Interest в однонаправленный, наша диаграмма должна слегка измениться. Пожалуйста, обратите внимание на то, что соответствующий элемент потока теперь имеет одну стрелку, тогда как у него было две стрелки: в начале и в конце. Сейчас он имеет только одну стрелку в конце, где идет соединение с накопителем Cash.

Теперь определите оставшиеся уравнения так, чтобы они выглядели такими же, как показано на рисунке 3.4. Обратите внимание на то, что вам не нужно вводить вызов функции максимума для сущности Net Interest. Он добавляется автоматически, так как поток объявлен однонаправленным. Что вам следует определить для потока, так это выражение `Cash * Interest_Rate`.

Те же самые уравнения могли быть непосредственно определены с помощью интегралов, но здесь мы использовали элементы потоковой диаграммы SFM. Накопитель SFM соответствует интегралу. Поток SFM связан с производной.

Если мы добавим элемент графика и отобразим на нем значение накопителя Cash способом, описанным в предыдущей главе 2, то тогда мы увидим следующий график, что демонстрирует рисунок 3.5. Здесь конечное время было увеличено до значения 36 в редакторе *Параметров моделирования*.

Идея заключается в том, что «ВизуальнаяАйвика» предоставляет разностороннюю поддержку уравнений, с помощью которых мы можем задавать и запускать достаточно сложные системы обыкновенных дифференциальных уравнений. Более того, «ВизуальнаяАйвика» име-

ет средства для проведения анализа чувствительности с помощью таких уравнений. В то же самое время «ВизуальнаяАйвика» позволяем нам моделировать и дискретно-событийные модели, которые могут быть скомбинированы с обыкновенными дифференциальными уравнениями.

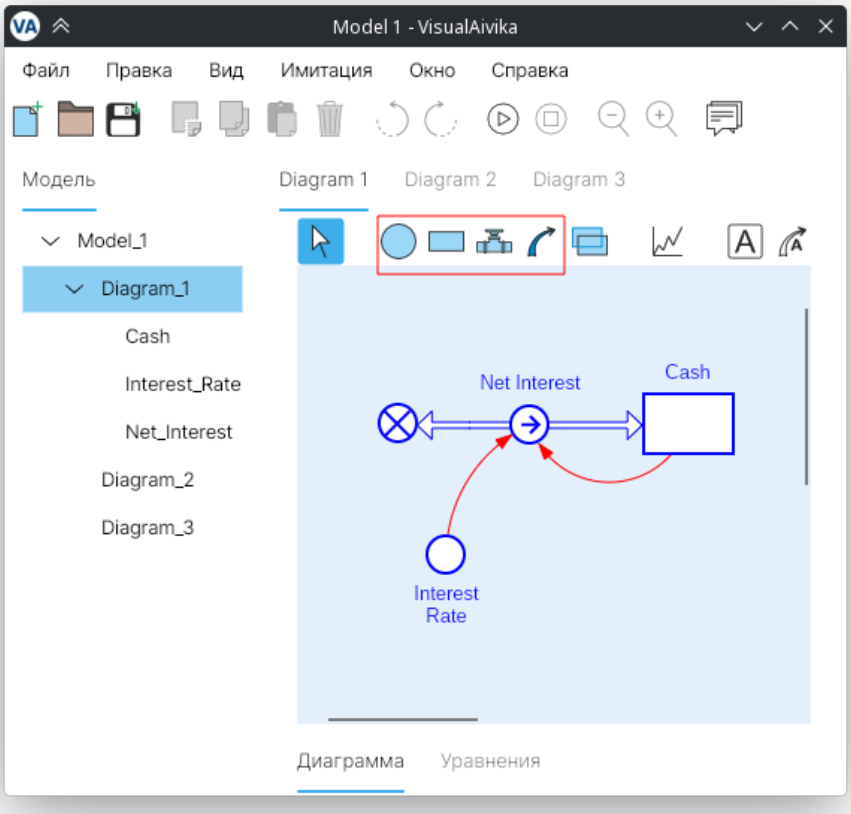


Рис. 3.1: Исходная потоковая диаграмма SFM для модели экспоненциального роста.

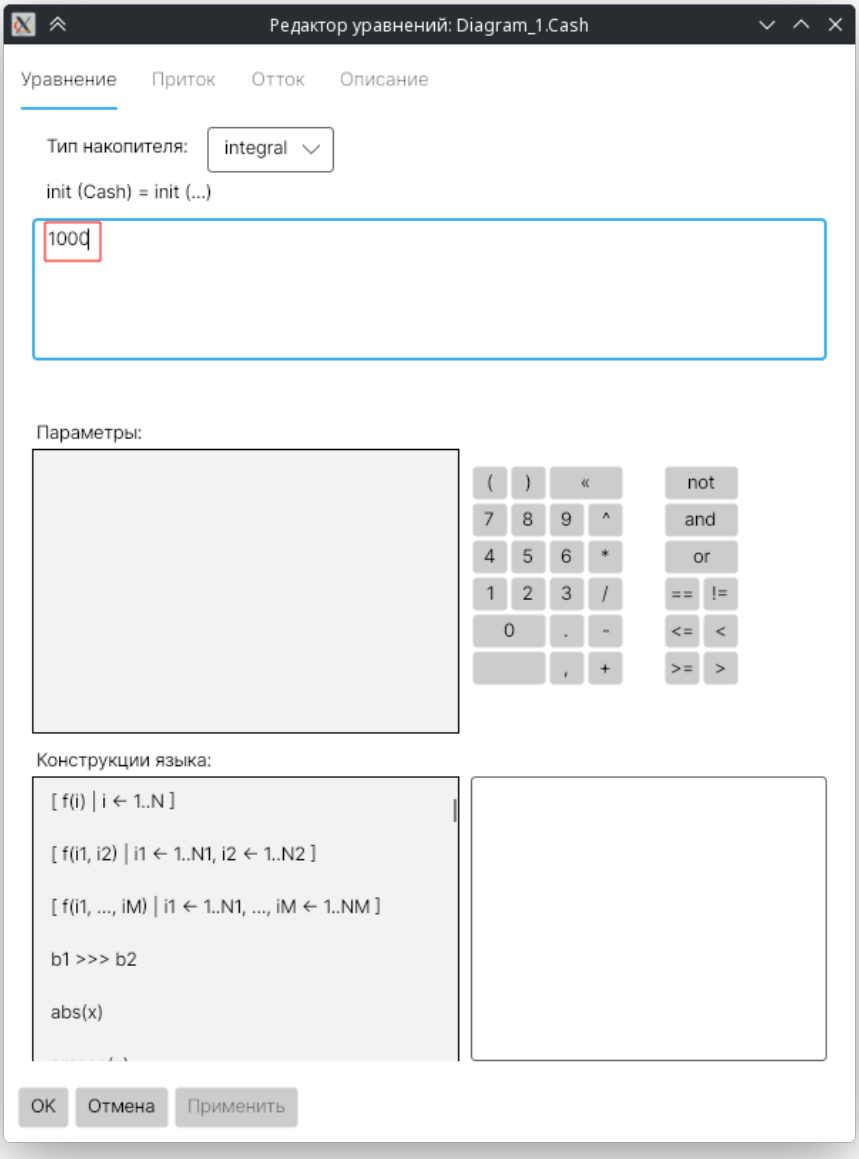


Рис. 3.2: Начальное значение накопителя SFM.

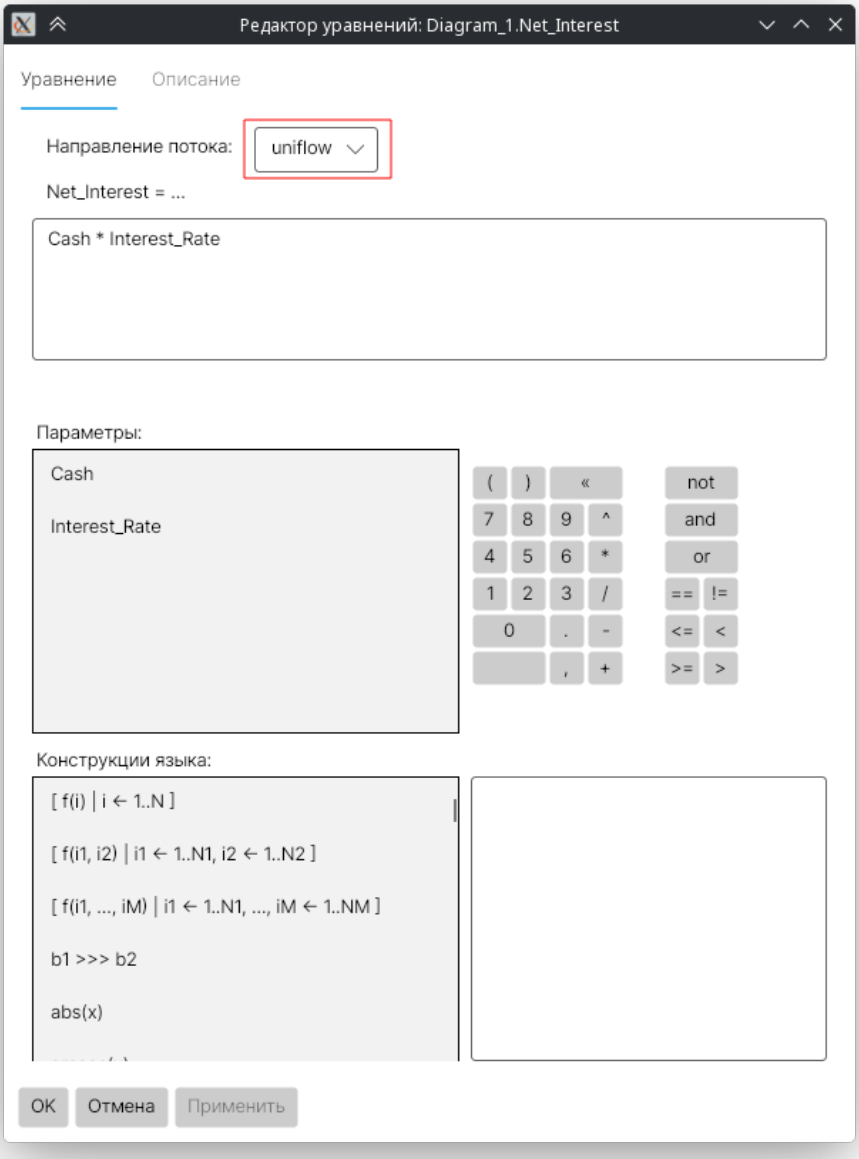


Рис. 3.3: Поток SFM становится однонаправленным.

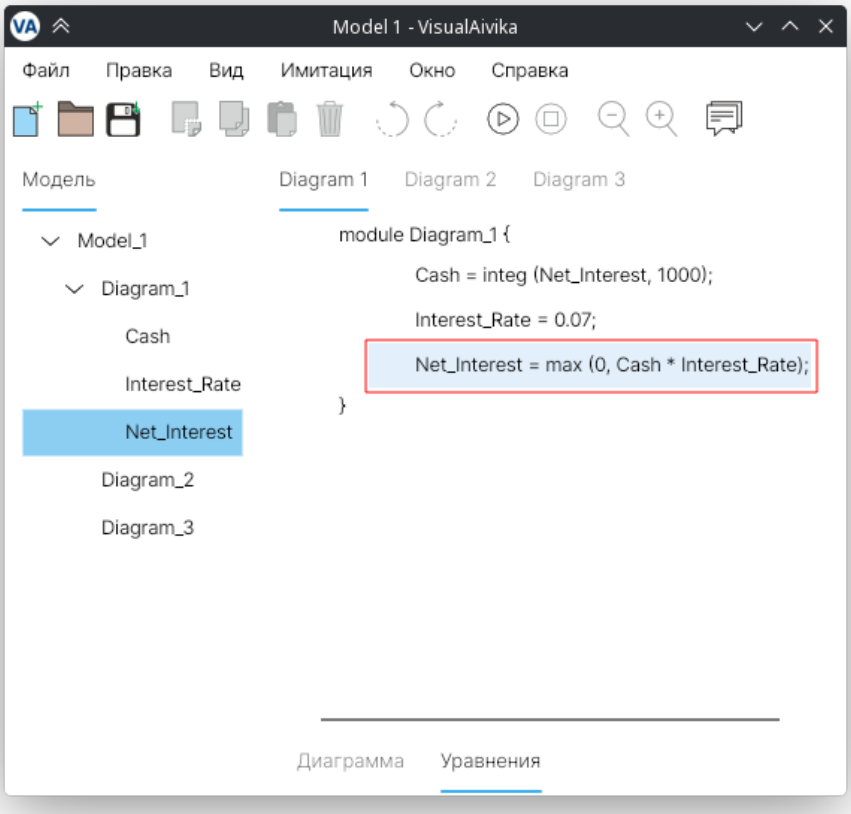


Рис. 3.4: Уравнения модели экспоненциального роста.

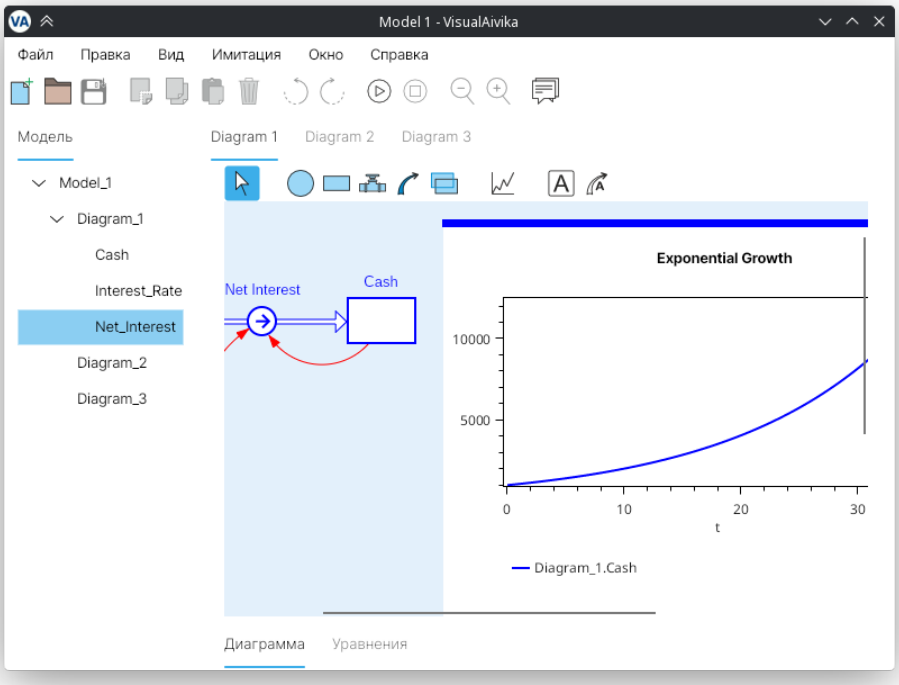


Рис. 3.5: График экспоненциального роста.

Глава 4

Как создать модель системы массового обслуживания

Для моделирования систем массового обслуживания «ВизуальнаяАйвика» использует подход похожий на язык моделирования GPSS[3], но реализация совершенная иная, которая имеет корни в функциональном программировании. Подобно GPSS система «ВизуальнаяАйвика» позволяет использовать блоки, но здесь блоки уже являются *вычислениями*, которые можно присваивать переменным подобно интегралам и числам.

На самом деле на момент написания этого текста в программе «ВизуальнаяАйвика» еще не было специальной визуальной поддержки блоков, но мы можем использовать те же самые дополнительные элементы SFM, которые мы использовали ранее в предыдущих главах. Мы просто присваиваем вычисления блоков переменным дополнительных элементов SFM. Как следствие, диаграмма состоит из элементов, соединенных в направлении противоположном тому, как на самом деле обрабатываются транзакты.

На рисунке 4.1 представлены блоки, которые примерно соответствуют следующему коду на языке GPSS.

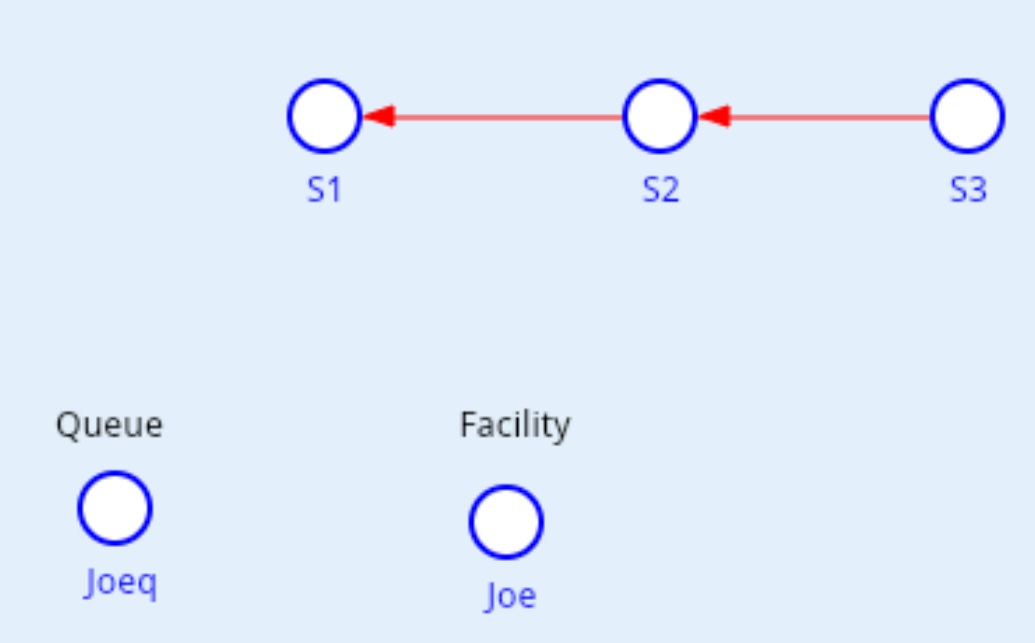


Рис. 4.1: Вычисления блоков как дополнительные элементы SFM.

Код на языке GPSS

```
GENERATE 18,6
QUEUE JOEQ
SEIZE JOE
DEPART JOEQ
ADVANCE 16,4
RELEASE JOE
TERMINATE

GENERATE 480
TERMINATE 1

START 1
```

Соответствующие уравнения в программе «ВизуальнаяАйвика» выглядят так, как показано на рисунке 4.2. Здесь сразу заметим, что «ВизуальнаяАйвика» проверяет соответствие диаграммы и уравнений. Оранжевым цветом указаны предупреждения, которые можно проигнорировать, потому что мы не хотим усложнять диаграмму.

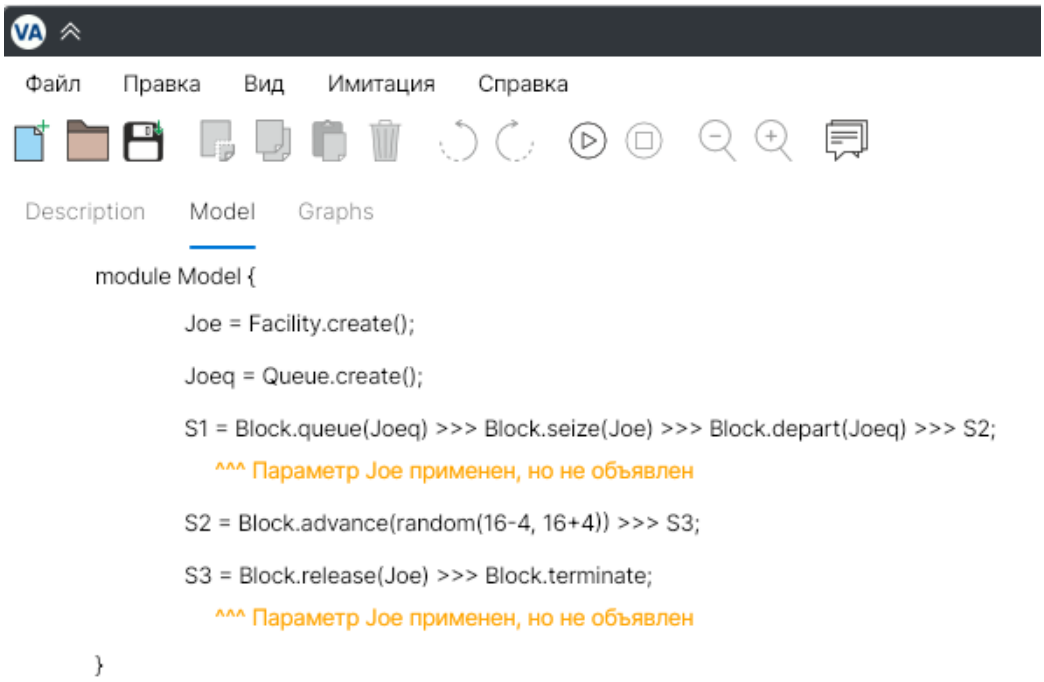


Рис. 4.2: Уравнения для вычислений блоков.

В программе «ВизуальнаяАйвика» блоки обладают свойством *композиционности*. Они могут быть соединены с другими блоками с помощью оператора >>> для создания новых блоков и цепочек блоков. Блок передает транзакты далее, тогда как цепочка блоков либо завершает обработку, либо создает бесконечный цикл.

Приведенные уравнения еще не закончены. Сейчас нам следует создать поток случайных событий и запустить всю имитацию. Для этого мы добавляем на диаграмму поток и соответствующий элемент запуска, как показано на рисунке 4.3.

Полные уравнения приведены на рисунке 4.4. Здесь мы заметим, что используется специальный оператор do! для запуска обработки транзактов по заданным потоку случайных событий и цепочке блоков. Действие запуска всегда явное в модели.

Теперь мы можем задать конечное время как 480 и сделать шаг интегрирования достаточно малым, скажем, равным 2,5.

На самом деле, шаг интегрирования не используется в модели си-

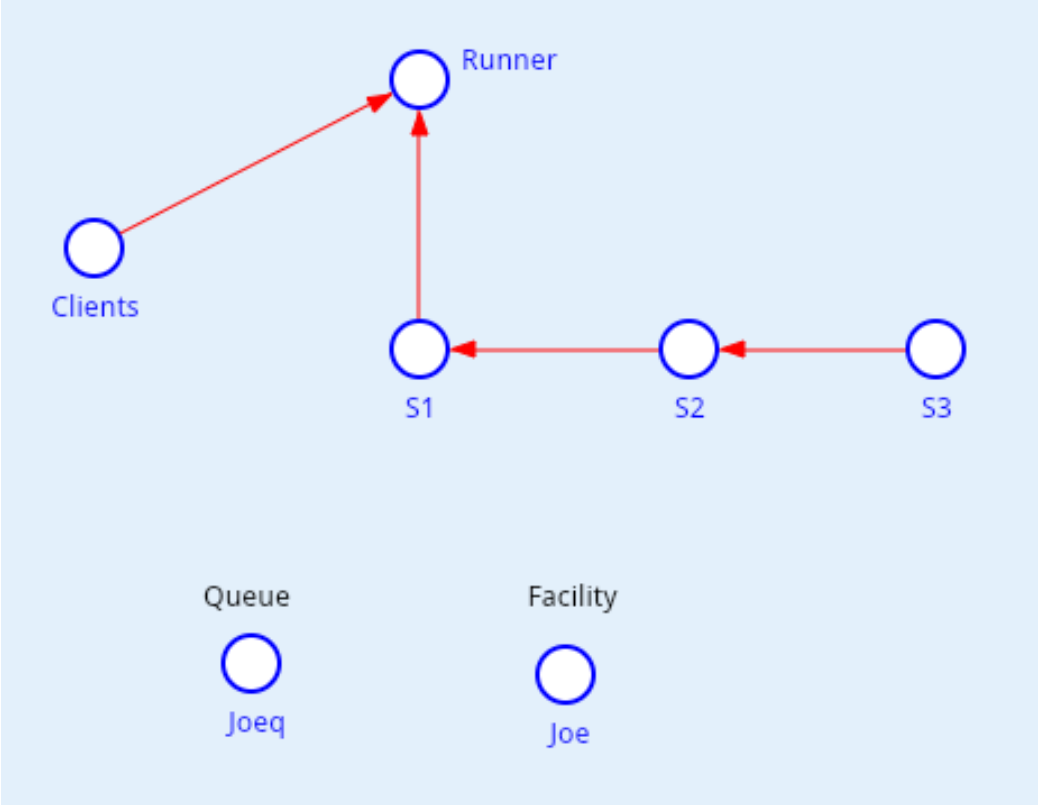


Рис. 4.3: Элементы завершенной модели.

стемы массового обслуживания, но он влияет на то, как рисуются графики, и как создаются таблицы с данными CSV. Графики рисуются всегда в точках интегрирования независимо от того, используется ли метод интегрирования или нет. Поэтому параметр `dt` должен иметь некоторое разумное значение.

Наконец пришло добавить некоторый вывод для результатов моделирования. Вы можете использовать главу 6 как справочник, где описаны разные методы отображения результатов. Кроме того, есть еще один элемент SFM, который может быть полезен сейчас, и который позволяет уменьшить нагромождение на визуальной диаграмме.

Добавьте к модели новую диаграмму, а затем создайте элемент *ссылки для SFM* на этой диаграмме, как показано на рисунке 4.5. Пусть эта ссылка будет связана с переменной `Joe` из главных уравнений

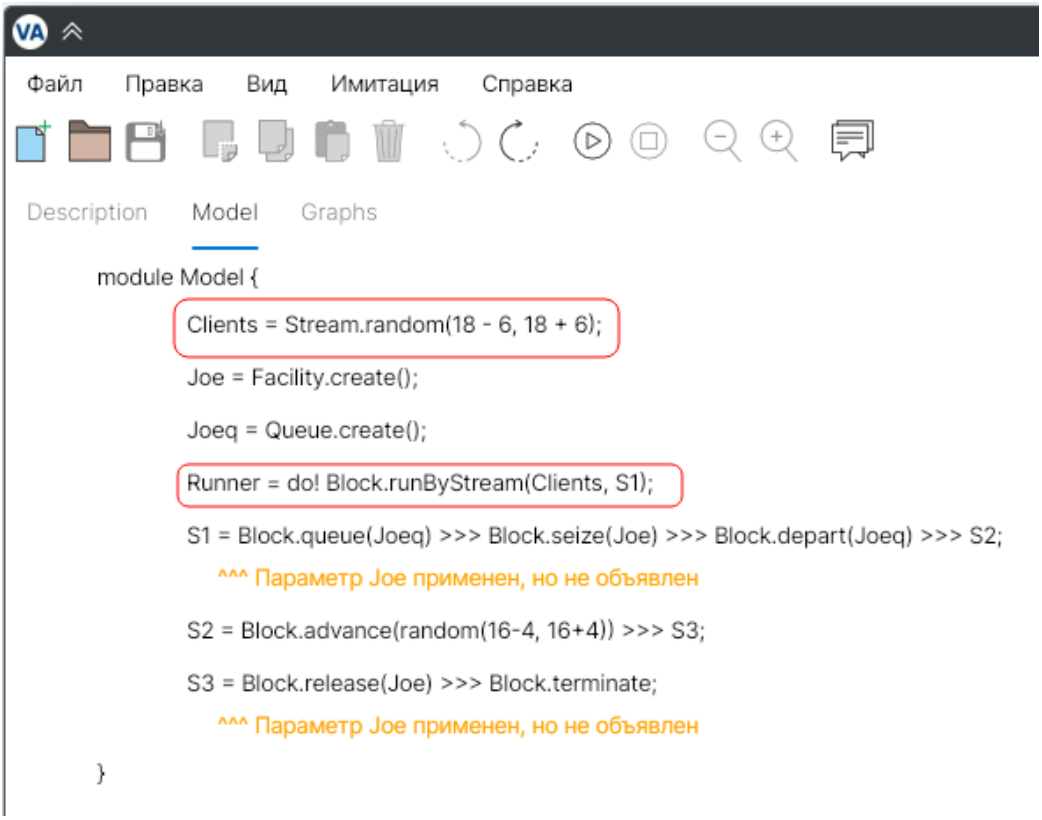


Рис. 4.4: Уравнения завершенной модели.

диаграммы `Model`. Также добавьте новые элементы, которые будут связанны с данным элементом ссылки SFM. Они будут возвращать свойства прибора, которые мы хотим увидеть на графиках.

Соответствующие уравнения для свойств приведены на рисунке 4.6. Использованные функции описаны в разделе 5.16.

Например, если мы нарисуем график отклонения для свойства длины очереди и его статистического аналога для 10000 имитационных запусков, то тогда мы можем получить следующие результаты, как показано на рисунке 4.7.

Обратите внимание на то, что оба графика отклонений сходятся, поскольку случайный процесс достаточно быстро становится стационарным[1]. Также мы не используем тот факт, что свойство неотри-

[1]Сброс статистики может быть добавлен в одной из следующих версий программы

цательно, но оно имеет достаточно широкий разброс из-за большого отклонения.

В этой модели мы не использовали обыкновенные дифференциальные уравнения, но могли бы. «ВизуальнаяАйвика» действительно позволяет нам использовать обыкновенные дифференциальные уравнения и дискретно-событийные части в одной комбинированной модели.

«ВизуальнаяАйвика» в будущем.

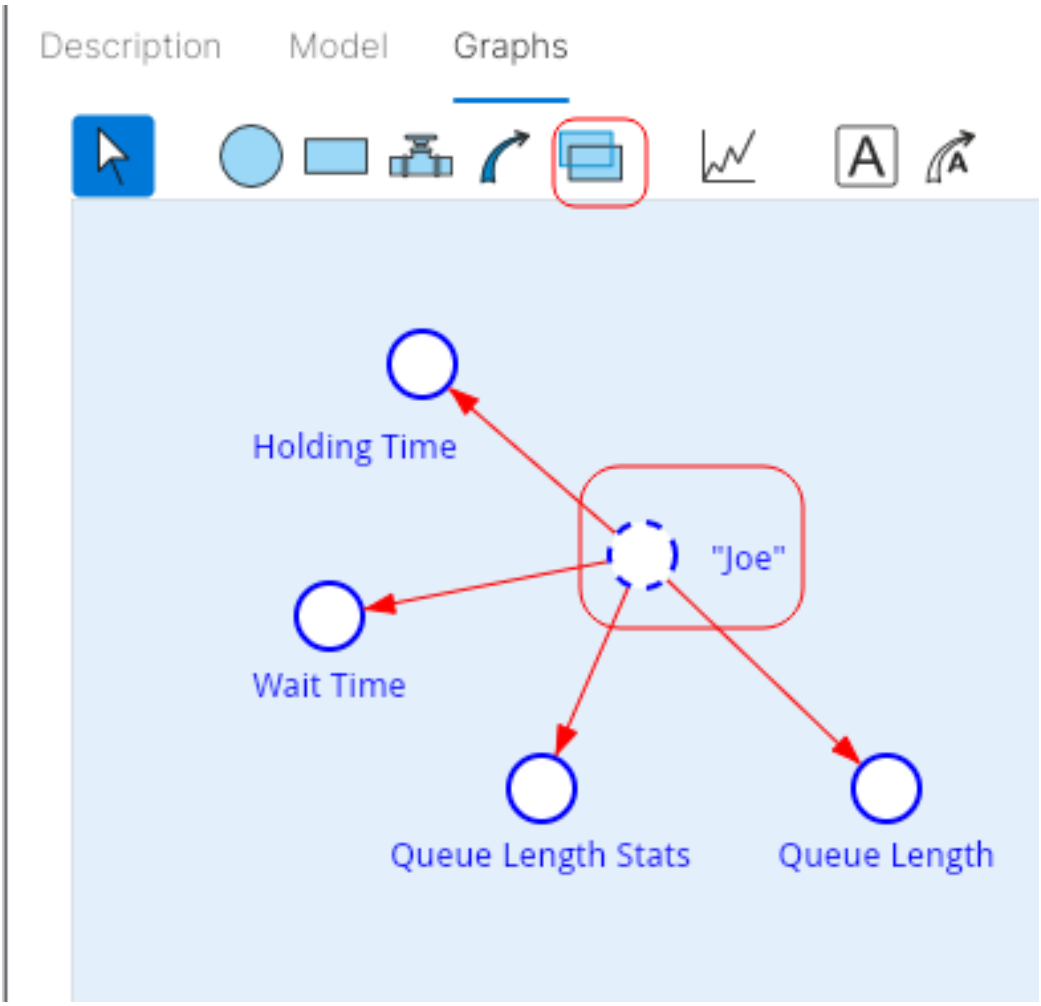


Рис. 4.5: Используйте ссылку SFM для уменьшения нагромождения на диаграммах.

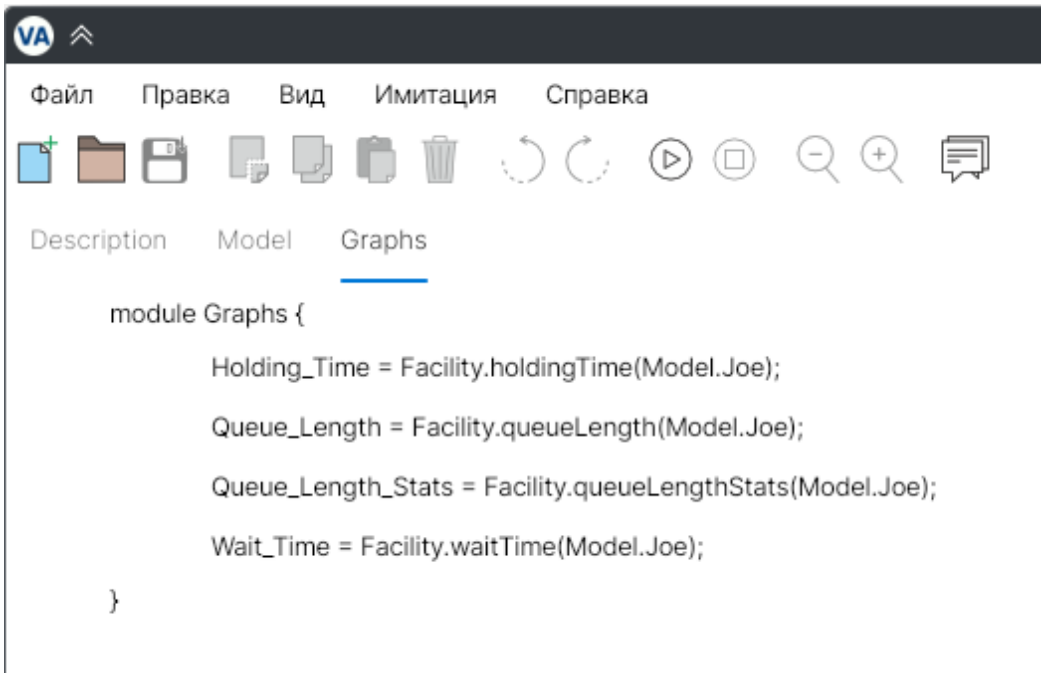


Рис. 4.6: Извлекаем свойства прибора.

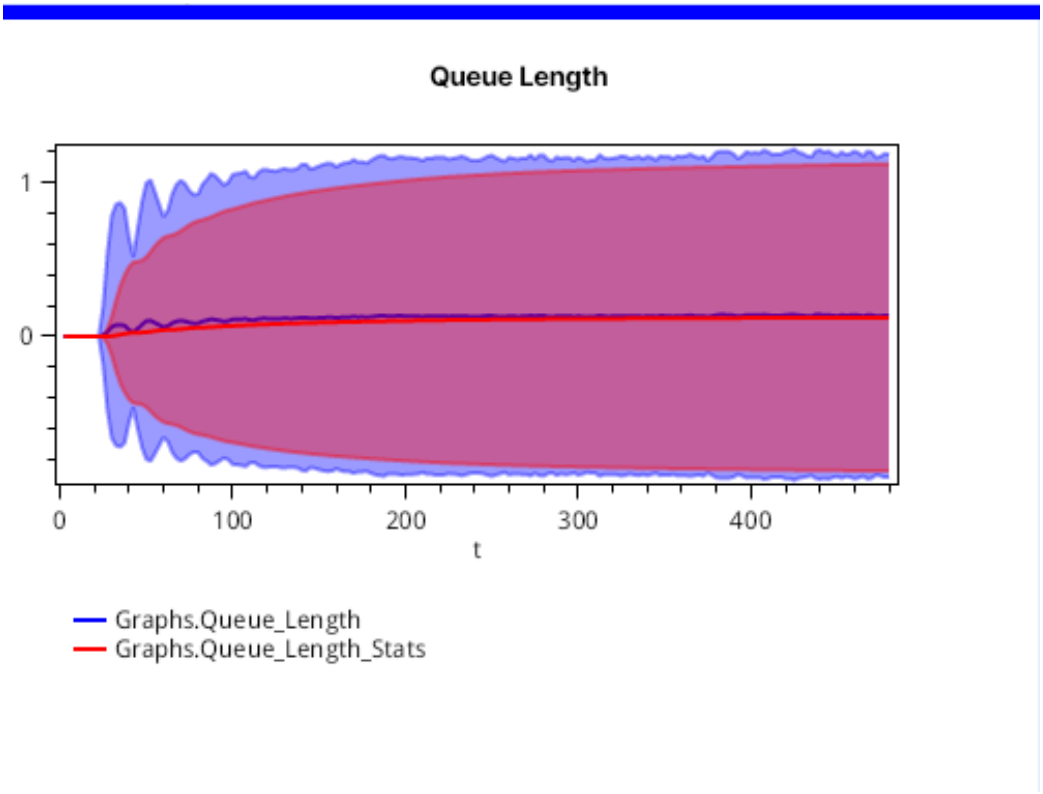


Рис. 4.7: Статистические результаты моделирования.

Глава 5

Язык моделирования

Язык моделирования программы «ВизуальнаяАйвика» позволяет вам задавать динамические системы. Модель может состоять из накопителей и дополнительных переменных. Переменная накопителя может быть интегралом (резервуаром). Дополнительная переменная — это функция от других переменных и вычислений.

В текущей версии сущности дискретно-событийного моделирования поддерживаются только через дополнительные переменные.

5.1 Параметры моделирования

Заданы предопределенные переменные, которые существуют в каждой имитации:

- `starttime` определяет начальное время;
- `stoptime` задает конечное время;
- `dt` определяет шаг интегрирования по модельному времени;
- `time` возвращает текущее модельное время.

Первые три константы подобно другим параметрам моделирования, описанным в этом разделе, могут быть определены в диалоговом окне, которое открывается через меню *Simulation / Simulation Specs*.

Если ваша имитационная модель содержит только сущности из области дискретно-событийного моделирования, то параметр `dt` может

быть задан любым. Однако его значение влияет на то, как отображаются графики, поскольку значения на графике дискретизируются с шагом `dt`.

«ВизуальнаяАйвика» поддерживает следующие методы интегрирования:

- метод Эйлера;
- метод Рунге-Кутта второго порядка;
- метод Рунге-Кутта четвертого порядка.

Также поддерживаются следующие режимы генерации случайных чисел:

- простой режим включает в себя стандартный генератор .NET для случайных чисел, который является слабым, но быстрым;
- строгий режим использует криптографический генератор случайных чисел, который возвращает более случайные числа, но он довольно медленный;
- режим с заданным начальным значением для генератора, который позволяет получить воспроизводимую последовательность псевдослучайных чисел для каждого имитационного запуска.

Более того, вы можете определить число запусков для вычислительного эксперимента по методу Монте-Карло, который отражается на следующих предопределенных переменных:

- `runIndex` возвращает индекс текущего запуска имитации, начиная с 0;
- `runCount` возвращает общее количество имитационных запусков в рамках заданного вычислительного эксперимента.

Эти две встроенные переменные полезны для планирования вычислительного эксперимента при проведении анализа чувствительности, как показано ниже в разделе 5.9.

5.2 Переменные

Следующий список определяет типы переменных в программе «ВизуальнаяАйвика».

- *Дополнительная переменная*. Это любая переменная, которая определена как функция от других переменных, или постоянная.
- *Накопитель*. Это динамическая переменная в модели. В текущей версии такая переменная может быть только резервуаром (интегралом).
- *Поток*. Это переменная, которая влияет на накопитель. Поток может быть либо входящим, либо исходящим. Также он может быть либо двунаправленным, либо однонаправленным.
- *Таблица*. Это табличная функция со значениями для координат x и y.
- *Диапазон*. Это диапазон индексов для массивов.
- *Блок*. Вычисление блока, которое обрабатывает транзакты и затем возвращает транзакты, либо те же самые, либо измененные.
- *Цепочка блоков*. Вычисление блока, которое обрабатывает транзакты, а затем либо завершает обработку, либо имеет бесконечный цикл.
- *Блок генератора*. Вычисление блока генератора, которое может начать обработку транзактов по заданной цепочке блоков.
- *Поток*. Поток входящих событий, которые приходят в моделируемую систему извне.
- *Сигнал*. Это дискретный сигнал, который может испускать входящие события.
- *Действие*. Некоторое действие, такое как запуск обработки транзактов по цепочке блоков.
- *Очередь*. Сущность очереди.

- *Прибор*. Прибор — это такой ресурс, который может иметь только одного владельца.
- *Многоканальное устройство*. Многоканальное устройство — это такой ресурс, который может быть использован несколькими транзактами.
- *Цепочка соответствия*. Цепочка соответствия может задерживать транзакты из одного и того же ансамбля.
- *Статистика по наблюдениям*. Значение статистики, которое основано на наблюдениях.
- *Статистика по времени*. Значение статистики, которое зависит от модельного времени.
- *Ссылка*. Изменяемая ссылка, ячейке которой могут быть присвоены числа и значения статистики.
- *Массив*. Массив из других переменных.

Название переменной в программе «ВизуальнаяАйвика» чувствительно к регистру. Первый символ должен быть буквой или подчеркиванием. Следующие символы могут содержать буквы, цифры и подчеркивания в любой последовательности.

Пример

```
Total_Resources, Scope, x, y
```

5.3 Уравнения

Язык моделирования программы «ВизуальнаяАйвика» позволяет вам определять модели, записывая множество математических уравнений и выражений. Уравнения могут быть записаны в любом порядке. Порядок вычислений определяется, исходя из зависимостей между переменными.

«ВизуальнаяАйвика» использует стандартные алгебраические выражения с теми же правилами приоритетов, которые используются в Java, C/C++ и других популярных языках программирования.

Также «ВизуальнаяАйвика» поддерживает стандартные математические функции, а также имеет дополнительные функции, которые делают процесс написание уравнений более быстрым и простым.

Пример

```
y = integ (1 + 0.2 * y * sin (t) - 1.5 * t ^ 2, 0);
z = integ ((y - z)^2, 0);
```

Здесь функция `integ` возвращает интеграл по заданной производной и начальному значению.

Пример

```
Adequacy_of_Control_Resources =
  Control_Resources / Desired_Control_Resources;
```

Здесь редактор уравнений *Equation Editor* автоматически завершает каждое уравнение символом точки с запятой.

5.4 Операторы

«ВизуальнаяАйвика» поддерживает стандартные двоичные и унарные операторы, которые следуют обычным правилам приоритетов. Операторы приведены в таблицах ниже.

Таблица 5.1: Унарные операторы

`not`	Логическое отрицание
`+ -`	Знак плюса и минуса

Таблица 5.2: Двоичные операторы

`^`	Возведение в степень
`* / + -`	Арифметические операторы
`< <= > >=`	Сравнение
`== !=`	Сравнение на равенство и неравенство

`and or`	Логическая конъюнкция и дизъюнкция с правилом короткого вычисления
`>>>`	Композиция блоков

Оператор композиции блоков требует некоторого разъяснения.

Выражение `b1 >>> b2` возвращает вычисление блока, которое обрабатывает транзакты первым блоком `b1`, а затем вторым блоком или цепочкой блоков `b2`. Если последний аргумент `b2` является блоком, то результат тоже будет блоком. В противном случае, если последний аргумент `b2` является цепочкой блоков, то тогда и результат будет также цепочкой блоков.

5.5 Постоянные

«ВизуальнаяАйвика» определяет следующие постоянные.

Таблица 5.3: Встроенные постоянные

`true`	Логическая ”истина”
`false`	Логическая ”ложь”
`pi`	Значение числа π
`infinity`	Представляет положительную бесконечность
`nan`	Представляет значение, которое не является числом

5.6 Функции

Ниже в таблице приведен список основных предопределенных функций.

Таблица 5.4: Основные функции

`abs (x)`	Возвращает абсолютное значение для `x`

`sqrt (x)`	Возвращает квадратный корень от `x`
`int (x)`	Возвращает наибольше целое число, которое меньше или равно, чем `x`
`round (x)`	Возвращает ближайшее число к `x`
`power (x, y)`	Возвращает то же самое, что и x^y
`mod (x, y)`	Возвращает остаток от деления `x/y`
`min (x1, x2, ...)`	Вовращает минимальное значение из `x1`, `x2`, ...
`max (x1, x2, ...)`	Возвращает максимальное значение из `x1`, `x2`, ...
`sum (x1, x2, ...)`	Возвращает сумму значений `x1`, `x2`, ...
`prod (x1, x2, ...)`	Возвращает произведение значений `x1`, `x2`, ...
`mean (x1, x2, ...)`	Возвращает среднее значение для `x1`, `x2`, ...
`sin (x)`	Возвращает значение синуса для `x`
`cos (x)`	Возвращает косинус для `x`
`tan (x)`	Возвращает тангенс для `x`
`arcsin (x)`	Возвращает арксинус для `x`
`arccos (x)`	Возвращает арккосинус для `x`
`arctan (x)`	Возвращает арктангенс для `x`
`arctan (y, x)`	Возвращает арктангенс для отношения `(y/x)`
`sinh (x)`	Возвращает гиперболический синус для `x`
`cosh (x)`	Возвращает гиперболический косинус для `x`
`tanh (x)`	Возвращает гиперболический тангенс для `x`
`sinwave (a, t)`	Возвращает синусоидальную волну с амплитудой `a` и периодом `t`
`coswave (a, t)`	Возвращает косинусоидальную волну с амплитудой `a` и периодом `t`
`exp (x)`	Возвращает e в степени `x`

`log (x)`	Возвращает натуральный логарифм от `x` (по основанию e)
`log (x, y)`	Возвращает логарифм от `x` по основанию `y`

Функции `sinwave` и `coswave` требуют некоторого пояснения. Они определены следующим образом.

Определение

```
sinwave(a, t) = a * sin(2 * pi * time / t);
coswave(a, t) = a * cos(2 * pi * time / t);
```

Таблица 5.5: Генераторы случайных чисел

`random (a, b)`	Возвращает равномерно распределенное случайное число между `a` и `b`
`randomInt (a, b)`	Возвращает равномерно распределенное целое случайное число между `a` и `b`
`triangular (a, m, b)`	Возвращает треугольно распределенное случайное число между `a` и `b` с медианой `m`
`normal (m, n)`	Возвращает нормально распределенное случайное число со средним `m` и отклонением `n`
`exponential (m)`	Возвращает показательно распределенное случайное число со средним `m`
`erlang (b, m)`	Возвращает случайное число по распределению Эрланга с масштабом `b` и целой формой `m`
`poisson (m)`	Возвращает случайное число по распределению Пуассона со средним `m`

`binomial (p, n)`	Возвращает биномиальное случайное число при количестве попыток `n` с вероятностью `p`

Таблица 5.6: Условное выражение

`if x then y else z`	Возвращает `y`, если `x` истинно; иначе возвращает `z`

Таблица 5.7: Различные функции

`step (h, t)`	Возвращает 0 до времени `t`, а затем возвращает `h`
`pulse (t, d)`	Возвращает импульс высоты 1, начиная со времени `t` и длительностью `d`
`pulse (t, d, p)`	Возвращает импульс высоты 1, начиная со времени `t`, длительностью `d` и периодом `p`
`ramp (s, t1, t2)`	Возвращает 0 до времени `t1`, а затем наклоняется ко времени `t2` с коэффициентом наклона `s`

Эти функции имеют следующее определение. Они используют описанный далее оператор `discrete`. Если кратко, этот оператор возвращает такое значение, которое не меняется за исключением интервалов интегрирования по dt независимо от того, каким бы ни был используемый метод интегрирования.

Определение

```
step(h, t) = discrete(if time + dt/2 > t then h else 0);

pulse(t, d) = discrete(if (time + dt/2 > t)
                          and (time + dt/2 < t + d)
                       then 1 else 0);

pulse(t, d, p) = pulse(t + (if (p > 0) and (time > t)
                            then round((time - t) / p)*p
```

```
                              else 0), d);

ramp(s, t1, t2) = discrete(if (time + dt/2 > t1)
                           then (if (time - dt/2 < t2)
                                 then s*(time - t1)
                                 else s*(t2 - t1))
                           else 0);
```

Таблица 5.8: Интерполяция

`table ((x1, y1), (x2, y2), ...)`	Создает таблицу, состоящую из точек `(x1, y1)`, `(x2, y2)`, ..., где таблица потом может быть использована как функция
`table ((x1, y1), (x2, y2), ...) (x)`	Оценивает значение `y` для заданного `x` по списку точек `(x1, y1)`, `(x2, y2)`, ..., используя линейную интерполяцию
`step (t)`	Создает функцию дискретной ступенчатой интерполяции на основе заданной таблицы `t`
`step (table ((x1, y1), (x2, y2), ...)) (x)`	Оценивает значение `y` для заданного `x` по списку точек `(x1, y1)`, `(x2, y2)`, ..., используя дискретную ступенчатую интерполяцию

Табличная функция интерполирует аргумент в соответствии с заданной таблицей.

Пример

```
Effect_of_Scope_Stability_of_Rules =
  table ((0, 0.5), (0.2, 0.535), (0.4, 0.585), (0.6, 0.675),
         (0.8, 0.82), (1, 1), (1.2, 1.21), (1.4, 1.35),
         (1.6, 1.42), (1.8, 1.47), (2, 1.5))
        (Scope);
```

Таблица 5.9: Функции интегралов

`integ (d, i)`	Возвращат интеграл с производной `d` и начальным значением `i`
`delay (x, t)`	Возвращает экспоненциальную задержку первого порядка `x` на время `t` с сохранением `x`
`delay (x, t, i)`	Возвращает экспоненциальную задержку первого порядка `x`, начиная с `i` и на время `t` с сохранением `x`
`delay3 (x, t)`	Возвращает экспоненциальную задержку третьего порядка `x` на время `t` с сохранением `x`
`delay3 (x, t, i)`	Возвращает экспоненциальную задержку третьего порядка `x`, начиная с `i` и на время `t` с сохранением `x`
`delayN (x, t, n)`	Возвращает экспоненциальную задержку `n`-го порядка `x` на время `t` с сохранением `x`
`delayN (x, t, n, i)`	Возвращает экспоненциальную задержку `n`-го порядка `x`, начиная с `i` и на время `t` с сохранением `x`
`smooth (x, t)`	Возвращает экспоненциальное сглаживание первого порядка `x` за время `t`
`smooth (x, t, i)`	Возвращает экспоненциальное сглаживание первого порядка `x` за время `t`, начиная с `i`
`smooth3 (x, t, i)`	Возвращает экспоненциальное сглаживание третьего порядка `x` за время `t`
`smooth3 (x, t, i)`	Возвращает экспоненциальное сглаживание третьего порядка `x` за время `t`, начиная с `i`
`smoothN (x, t, n)`	Возвращает экспоненциальное сглаживание `n`-го порядка `x` за время `t`

`smoothN (x, t, n, i)`	Возвращает экспоненциальное сглаживание `n`-го порядка `x` за время `t`, начиная с `i`
`forecast (x, t, h)`	Прогноз для `x` на временном горизонте `h` с использованием среднего значения времени `t`
`trend (x, t, i)`	Возвращает дробную скорость изменения `x`, используя среднее время `t` и начиная с `i`

Здесь функции из семейства `delay` используют следующую идею, где функции `delayN` являются обобщением правила. Если вы будете сравнивать эти функции с другими программными инструментами моделирования, то имейте в виду, что функции `delayN` не имеют эффекта оператора `discrete`, который неявно может присутствовать в других инструментах моделирования. В случае необходимости этот оператор может быть вручную добавлен в уравнения.

Определение

```
D1=delay(x, t) то же самое, что и
  D1=1/t * integ(x - D1, x*t);

D1I=delay(x, t, i) то же самое, что и
  D1I=1/t * integ(x - D1I, i*t);

D3_3=delay3(x, t) то же самое, что и
  D3_3=1/(t/3) * integ(D3_2 - D3_3, x*t/3);
  D3_2=1/(t/3) * integ(D3_1 - D3_2, x*t/3);
  D3_1=1/(t/3) * integ(x - D3_1, x*t/3);

D3I_3=delay3(x, t, i) то же самое, что и
  D3I_3=1/(t/3) * integ(D3I_2 - D3I_3, i*t/3);
  D3I_2=1/(t/3) * integ(D3I_1 - D3I_2, i*t/3);
  D3I_1=1/(t/3) * integ(x - D3I_1, i*t/3);
```

Например, рисунок 5.1 показывает функции задержки первого и третьего порядка для следующего входа и временного интервала.

Пример

```
x=sin(time);
t=40*dt;
dt=0.025;
```

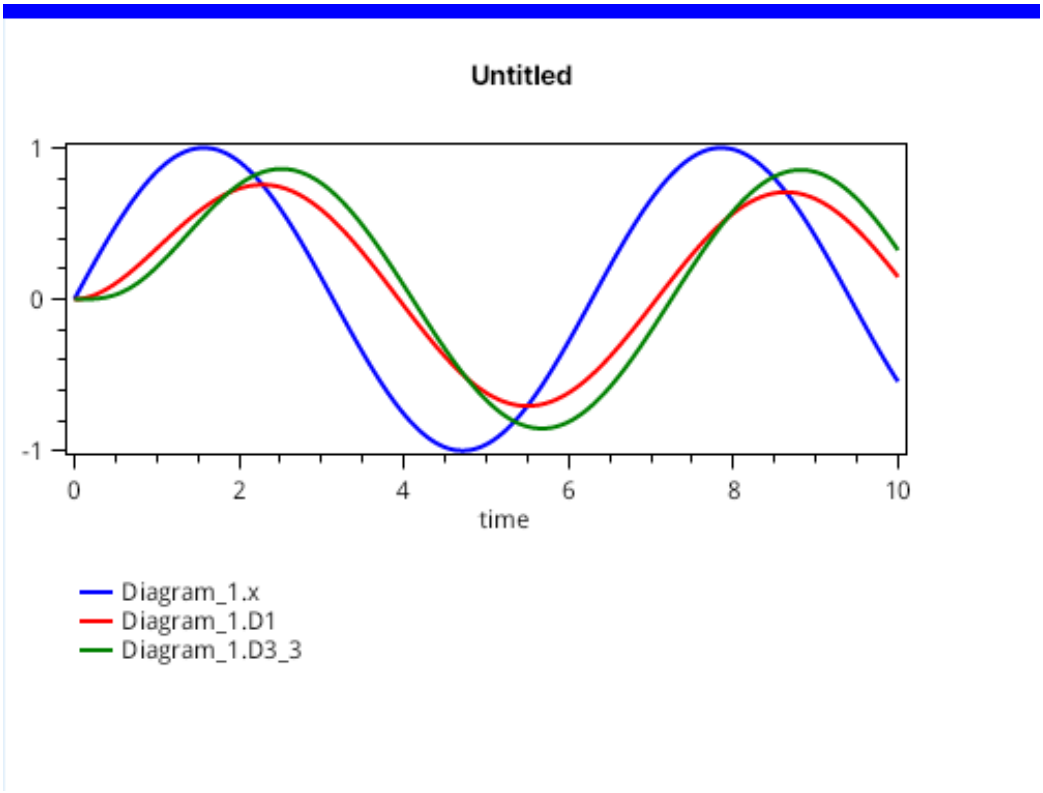


Рис. 5.1: Задержка первого порядка, `D1`, и задержка третьего порядка, `D3_3`, для заданного входа, `x`, и периода времени.

Функции из семейства `smooth` используют похожую идею, где функции `smoothN` являются обобщением правила. Если вы будете сравнивать эти функции с другими программными инструментами моделирования, то имейте в виду, что функции `smoothN` не имеют эффекта оператора `discrete`, который неявно может присутствовать в других инструментах моделирования. В случае необходимости этот оператор может быть вручную добавлен в уравнения.

Определение

```
S1=smooth(x, t) то же самое, что и
  S1=integ((x - S1)/t, x);

S1I=smooth(x, t, i) то же самое, что и
  S1I=integ((x - S1I)/t, i);

S3_3=smooth3(x, t) то же самое, что и
  S3_3=integ((S3_2 - S3_3)/(t/3), x);
  S3_2=integ((S3_1 - S3_2)/(t/3), x);
  S3_1=integ((x - S3_1)/(t/3), x);

S3I_3=smooth3(x, t, i) то же самое, что и
  S3I_3=integ((S3I_2 - S3I_3)/(t/3), i);
  S3I_2=integ((S3I_1 - S3I_2)/(t/3), i);
  S3I_1=integ((x - S3I_1)/(t/3), i);
```

Функции `delay` и `smooth` ведут себя по-разному, когда период времени начинает меняться.

Функции `forecast` и `trend` имеют следующее определение, где оператор `init` возвращает значение выражения в начальной точке моделирования.

Определение

```
forecast(x, t, h) = x * (1 + (x / smooth(x, t) - 1) / t * h);

trend(x, t, i) = (x / smooth(x, t,
                   init(x) / (1 + i * init(t))) - 1) / t;
```

Таблица 5.10: Финансовые функции

`npv (s, r, i, f)`	Возвращает чистую приведенную стоимость (NPV) потока `s`, вычисленную по заданной ставке дисконта `r`, начальному значению `i` и некоторому коэффициенту `f` (обычно 1)

`npve (s, r, i, f)`	Возвращает конечную чистую приведенную стоимость (NPVE) потока `s`, вычисленную по заданной ставке дисконта `r`, начальному значению `i` и некоторому коэффициенту `f`

В программе «ВизуальнаяАйвика» финансовые функции имеют следующее определение.

Определение

```
npv=npv(s, r, i, f) то же самое, что и

  npv = (accum + dt * s * df) * f;
  df = integ(- df * r, 1);
  accum = integ(s * df, i);

npve=npve(s, r, i, f) то же самое, что и

  npve = (accum + dt * s * df) * f;
  df = integ(- df * r / beta, 1 / beta);
  accum = integ(s * df, i);
  beta = 1 + r * dt;
```

Таблица 5.11: Операторы моделирования

`discrete (x)`	Возвращает значение `x`, которое не меняется за исключением интервалов интегрирования `dt` независимо от того, каким бы ни был используемый метод интегрирования
`init (x)`	Возвращает значение выражения `x` в начальной точке моделирования

Оператор `init` возвращает начальное значение для интеграла. Также он возвращает блок `Block.identity` для любого блока, также как возвращает `Block.terminate` для любой цепочки блоков. Наконец, оператор возвращает вычисление `Stream.empty` для любого потока событий.

В отличие от других программных инструментов моделирования «ВизуальнаяАйвика» поддерживает довольно нестандартные операторы `discrete` и `init`. Они связаны с самим движком моделирования программы «ВизуальнаяАйвика». Любое числовое выражение может иметь начальное значение. Также мы можем трактовать любое числовое выражение как нечто, что менялось бы, как если бы использовался метод Эйлера. Обычно вам не следует применять эти операторы в ваших моделях. Более того, последние два оператора могут быть специфичными для программы «ВизуальнаяАйвика», и они могут быть плохо переносимыми между разными программными инструментами моделирования.

5.7 Диапазоны

Диапазон задается двумя целочисленными выражениями: нижний индекс диапазона и его верхний индекс. Выражения должны быть ограничены двумя точками.

Пример

```
N = 10;

I_Range = 1..N;
J_Range = 1..5;
```

Диапазоны могут быть заданы на диаграмме подобно другим переменным, а затем использованы в уравнениях. Они нужны для массивов.

5.8 Массивы

Для определения одномерного массива вы можете использовать особый синтаксис:

[*Element* | *Index* <- *Range*]

Здесь выражение элемента может зависеть от индекса, чьи значения будут получены из заданного диапазона.

Синтаксис обобщен также для других размерностей. Например, двумерный массив может быть определен после добавления другого индекса:

[*Element* | *Index1* <- *Range1* , *Index2* <- *Range2*]

«ВизуальнаяАйвика» поддерживает до 5 размерностей.

Чтобы сослаться на элемент одномерного массива по заданному индексу, вы можете использовать следующую конструкцию:

Array[*Index*]

Похожим образом можно ссылаться на элементы двумерного массива по двум индексам:

Array2D[*Index1* , *Index2*]

Это правило также обобщается на случай других размерностей.

Пример

```
n = 51;

C = [ if (i == 0) or (i == n + 1) then 0 else M[i] / v |
      i <- 0..n+1 ];

M = [ integ (q + k*(C[i-1] - C[i]) + k*(C[i + 1] - C[i]), 0) |
      i <- 1..n ];

q = 1;
k = 2;
v = 0.75;
```

В этом примере C и M являются одномерными массивами с разными индексами. Массив C имеет значения `C[0]`, ..., `C[n+1]`, тогда как массив M имеет значения `M[1]`, ..., `M[n]`.

Между прочим, диапазоны могли быть определены как отдельные переменные. Для простоты здесь диапазоны определены внутри массивов. Применимы оба метода.

5.8.1 Функции для массивов и диапазонов

Для массивов и диапазонов определены следующие функции.

Таблица 5.12: Функции для массивов и диапазонов

`length (a)`	Возвращает общее количество элементов для заданного диапазона или массива `a`
`length (a, d)`	Возвращает количество элементов для заданного массива `a` и размерности `d`, начиная с 0
`low (a)`	Возвращает нижний индекс для заданного диапазона или одномерного массива `a`
`low (a, d)`	Возвращает нижний индекс для заданного массива `a` и размерности `d`, начиная с 0
`high (a)`	Возвращает верхний индекс для заданного диапазона или одномерного массива `a`
`high (a, d)`	Возвращает верхний индекс для заданного массива `a` и размерности `d`, начиная с 0
`min (a)`	Возвращает минимальный элемент заданного массива `a`
`max (a)`	Возвращает максимальный элемент заданного массива `a`
`sum (a)`	Возвращает сумму элементов для заданного массива `a`
`prod (a)`	Возвращает произведение элементов для заданного массива `a`
`mean (a)`	Возвращает среднее значение для элементов заданного массива `a`

Последние функции являются агрегирующими. Полезно, что мы можем передать таким функциям временные массивы, которые сами являются результатом выражений.

Следующий пример эквивалентен примеру из [Vensim 5 Reference Manual, страница 30, пример 3] документации к Vensim[5].

Пример

```
efficiency = prod(factor_efficiency);
```

```
US_population = sum(population);
revenue = [ sum([ sales[c, p] * price[p, b] | p <- product ]) |
              c <- country, b <- brand ];
```

Пожалуйста, обратите внимание на то, как создается временный массив для суммирования элементов в массиве-переменной revenue.

Следующий пример относится к теории игр. Он вычисляет значения максимина и минимакса по заданной матрице размерности $n \times n$, соответственно.

Пример

```
max_min = max([ min([ A[i,j] | j <- 1..n ]) | i <- 1..n ])
min_max = min([ max([ A[i,j] | j <- 1..n ]) | i <- 1..n ])
```

Здесь временные массивы создаются только один раз в самом начале запуска имитации.

Массивы могут содержать и вычисления блоков, и ресурсы с очередями. Тогда к ним можно обращаться по индексу.

5.8.2 Инициализация массива

Некоторые массивы могут быть заданы в табличной форме без явного указания диапазонов и индексов. Существует упрощенный синтаксис для этого.

Пример

```
A1 = [0, 1, 2, 3, 4, 5];
A2 = [[0, 1], [2, 3], [4, 5]];
A3 = [[[0, 1], [2, 3]], [[4, 5], [6, 7]]];
```

Только такие массивы всегда имеют индексы, начинающиеся с 0.

5.8.3 Отображение массивов на графиках

Массивы могут быть отображены на графиках. Например, определенный через агрегирование в разделе 5.8.1 массив revenue может выглядеть как на графике 5.2.

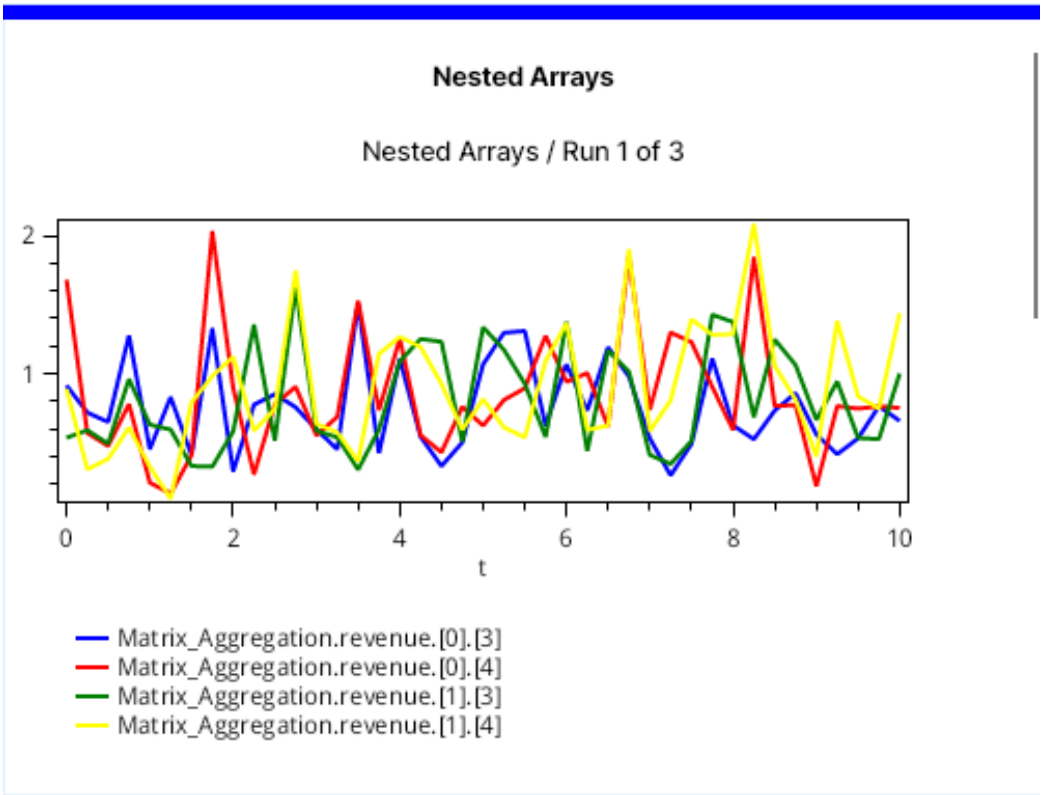


Рис. 5.2: При отображении массива на графике каждый элемент имеет свой индекс.

5.9 Анализ чувствительности

Как было замечено выше в тексте, существуют встроенные переменные `runIndex` и `runCount`, которые возвращают индекс текущего запуска, начиная с 0, и общее количество запусков в рамках одного вычислительного эксперимента по методу Монте-Карло, соответственно.

Важно, что переменная `runIndex` является постоянной в рамках текущего имитационного запуска, а затем обновляется для другого запуска. Существуют похожие случайные параметры с тем же свойством. Они постоянны в рамках текущего запуска, а затем они переопределяются для другого запуска.

Таблица 5.13: Случайные параметры

`randomParam (a, b)`	Возвращает равномерно распределенный случайный параметр между `a` и `b`
`randomIntParam (a, b)`	Возвращает равномерно распределенный целый случайный параметр между `a` и `b`
`triangularParam (a, m, b)`	Возвращает треугольно распределенный случайный параметр между `a` и `b` с медианой `m`
`normalParam (m, n)`	Возвращает нормально распределенный случайный параметр со средним `m` и отклонением `n`
`exponentialParam (m)`	Возвращает показательно распределенный случайный параметр со средним `m`
`erlangParam (b, m)`	Возвращает случайный параметр по распределению Эрланга с масштабом `b` и целой формой `m`
`poissonParam (m)`	Возвращает случайный параметр по распределению Пуассона со средним `m`
`binomialParam (p, n)`	Возвращает биномиальный случайный параметр при количестве попыток `n` с вероятностью `p`

Такие параметры полезны для проведения анализа чувствительности. Они могут быть использованы в уравнениях.

Переопределяя некоторые постоянные как случайные внешние параметры, мы можем проверить модель на устойчивость. Для этого «ВизуальнаяАйвика» поддерживает метод Монте-Карло. Результаты такого анализа могут быть отображены на графике подобно рисунку 5.3, где используется опция *Deviation Chart* для элемента *Graph Element*.

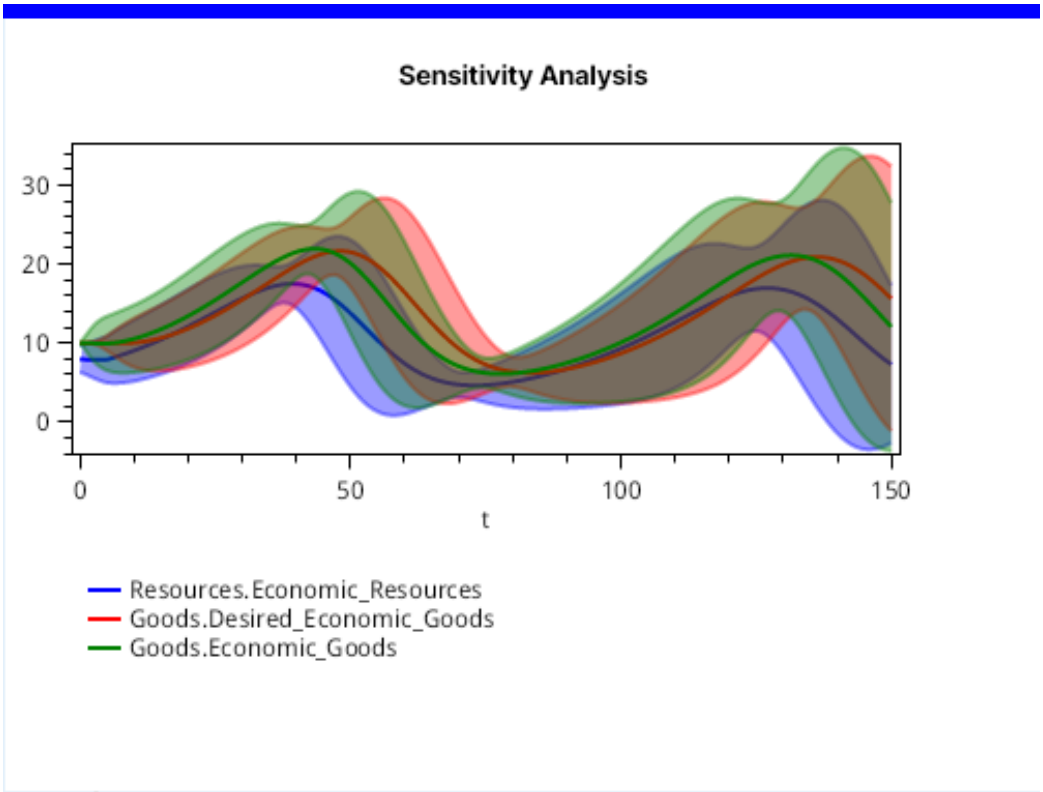


Рис. 5.3: График показывает тренды и доверительные интервалы, т.е. насколько устойчива модель относительно изменений внешних случайных параметров.

Более того, сочетая инициализацию массивов со встроенными переменными `runIndex` и `runCount`, мы можем задать выборку для внешних параметров. Идея довольно проста. Мы определяем массив со значениями, а затем ссылаемся на него по индексу запуска, вероятно, ограниченному размером массива.

Пример

```
A = [[1, 2, 3],
     [2, 3, 1],
     [3, 1, 2]];

Sample = mod(runIndex, length(A, 0));
```

```
X1 = A[Sample, 0];
X2 = A[Sample, 1];
X3 = A[Sample, 2];
```

5.10 Транзакты

Вычисления блоков обрабатывают транзакты. Большинство блоков принимает выражения, которые могут иметь доступ к атрибутам транзакта по названию, которое может быть произвольным идентификатором.

Таблица 5.14: Примеры атрибутов транзактов

`transact.attribute`	Обращение к атрибуту «attribute» транзакта
`transact.ABC`	Обращение к атрибуту «ABC» транзакта

Транзакт может иметь произвольное число атрибутов.

При разделении транзакта на копии, изменения атрибутов транзакта не влияют на другие транзакты из того же ансамбля.

5.11 Поток внешних событий

Перед созданием транзактов в блоке генератора в имитационную модель извне должны прийти соответствующие внешние события. Такие события определяются с помощью вычисления потока `Stream`.

Таблица 5.15: Вычисление Stream

`Stream.empty`	Возвращает пустой поток, который не создает внешних событий
`Stream.take(s, n)`	Взять заданное количество `n` внешних событий из потока `s`

`Stream.random(a, b)`	Возвращает новый поток внешних событий с равномерным распределением случайных задержек между `a` и `b`
`Stream.randomInt(a, b)`	Возвращает новый поток внешних событий с равномерным распределением целочисленных случайных задержек между `a` и `b`
`Stream.triangular(a, m, b)`	Возвращает новый поток внешних событий с треугольным распределением случайных задержек между `a` и `b` с медианой `m`
`Stream.normal(m, n)`	Возвращает новый поток внешних событий с нормальным распределением случайных задержек со средним `m` и отклонением `n`
`Stream.exponential(m)`	Возвращает новый поток внешних событий с показательным распределением случайных задержек со средним `m`
`Stream.erlang(b, m)`	Возвращает новый поток внешних событий, где случайные задержки распределены по закону Эрланга с масштабом `b` и целой формой `m`
`Stream.poisson(m)`	Возвращает новый поток внешних событий, где случайные задержки распределены по закону Пуассона со средним `m`

`Stream.binomial(p, n)`	Возвращает новый поток внешних событий с биномиальным распределением случайных задержек при количестве попыток `n` с вероятностью `p`

Чтобы создать заданное число внешних событий, скажем 15, в начальное время имитации, мы можем создать поток по шаблону следующего примера.

Пример

```
S = Stream.take(Stream.random(0, 0), 15);
```

Здесь мы создаем бесконечный поток, а затем отбираем только первые 15 событий из него.

5.12 Блок генератора

Вычисление блока генератора такое, как `GeneratorBlock.byStream`, может принять поток внешних событий, цепочку блоков, а затем начать создавать и обрабатывать транзакты. Обычно это происходит неявно, когда применяется функция `Block.runByStream`, но информацию о блоках генератора все же следует привести в этом документе.

Таблица 5.16: Блок генератора

`GeneratorBlock.byStream(s)`	Возвращает блок генератора по заданному потоку `s` внешних событий

`GeneratorBlock.bySignal(s)`	Возвращает блок генератора по заданному сигналу `s`, который возбуждает внешние события

Обе эти функции похожи. Только первая принимает поток, тогда как вторая функция принимает сигнал. Сигналы рассматриваются в разделе 5.20.

5.13 Блоки

Обработка транзактов происходит внутри блоков. Блок может задерживать транзакты, может менять их, а затем разрушать. Блоки можно соединять с помощью композиции в том смысле, что мы можем создать цепочку блоков, которая даже может быть бесконечным циклом, или она может иметь обратные связи в случае необходимости.

Оператор композиции выглядит как `b1 >>> b2`, где транзакты сначала обрабатываются блоком `b1`, а затем блоком или цепочкой блоков `b2`.

Разница между обычным блоком и цепочкой блоков состоит в том, что цепочка блоков либо завершает обработку, либо имеет бесконечный цикл. С другой стороны, блок передает транзакт дальше, возможно, меняя один из его атрибутов.

Мы можем вводить ветвление вычислений блоков с помощью оператора `Block.select`, или мы можем соединять несколько блоков в одно вычисление с помощью оператора композиции.

Композиция блоков противоположна тому направлению, в котором обрабатываются транзакты!

Таблица 5.17: Основные вычисления блоков

`Block.select(if x then y else z)`	Возвращает цепочку блоков, которая обрабатывает транзакты с помощью цепочки блоков `y`, если условие `x` выполняется, иначе транзакты обрабатываются с помощью цепочки блоков `z`
`Block.terminate`	Возвращает цепочку блоков, которая завершает обработку транзактов
`Block.identity`	Возвращает блок, который просто передает входной транзакт дальше
`Block.transfer (b)`	Возвращает цепочку блоков, которая перенаправляет обработку транзактов в заданную цепочку блоков `b`. Это позволяет создавать обратные связи
`Block.assign (transact.attribute <- v)`	Возвращает блок, который назначает заданному атрибуту транзакта значение `v` при обработке транзакта
`Block.advance(v)`	Возвращает блок, который задерживает транзакт на заданный интервал времени `v`, который может быть выражением, которое может зависеть от атрибутов транзакта

`Block.priority(p)`	Возвращает блок, который назначает новый приоритет транзактам, где `p` может быть выражением, которое может зависеть от атрибутов транзакта

Следующий пример создает цепочку блоков, которая задерживает каждый транзакт на 10 единиц времени, затем задерживает на интервал времени, который вычисляется из выражения, а потом, наконец, завершает обработку.

Пример

```
B = Block.advance(10) >>>
      Block.advance(20 + transact.SomeDelay) >>>
      Block.terminate;
```

Ниже приведен другой пример, который определяет простейший бесконечный цикл. Функция `Block.transfer` позволяет симулятору разорвать циклическую зависимость. Иначе уравнение не могло бы разрешиться.

Пример

```
B = Block.transfer(B);
```

Мы можем использовать следующий пример для сохранения текущего модельного времени в произвольном атрибуте `ArrivalTime`. Затем транзакт передается заданной цепочке блоков.

Пример

```
B = Block.assign(transact.ArrivalTime <- time) >>> B2;
```

Ниже описана небольшая уловка. Если цепочка блоков состоит из множества блоков, то имеет смысл назвать их с помощью одних и тех же идентификаторов, которые будут отличаться по окончанию, которое уже будет указывать на номер шага.

Пример

```
B1 = Block.advance(10) >>> B2;
B2 = Block.advance(20) >>> B3;
B3 = Block.advance(30) >>> B4;
B4 = Block.terminate;
```

Тогда такие уравнения появятся на вкладке *Equations* последовательно друг за другом. Уравнения сортируются по названию.

Наконец, следующий пример показывает, как создать две ветви B2 и B3 из одного вычисления, где решение будет зависеть от генератора случайных чисел. Здесь мы могли бы, например, проверить доступные ресурсы.

Пример

```
B = Block.select(if random(0, 1) < 0.3 then B2 else B3);
```

Напротив, если у нас есть два вычисления блоков B4 и B5, то мы можем направить оба из них в ту же самую цепочку блоков B, как показано ниже.

Пример

```
B8 = B4 >>> B;
B9 = B5 >>> B;
```

Тогда оба пути обработки транзактов сольются в один путь.

5.14 Запуск блоков

При наличии некоторой цепочки блоков мы можем запустить обработку транзактов с помощью этой цепочки блоков. Для этого нам нужен блок генератора, который может быть создан с помощью потока внешних событий или с помощью сигнала о таких событиях.

Таблица 5.18: Запуск вычисления Block

`Block.run(g, b)`	Возвращает действие, которое может быть применено к оператору do! для запуска обработки транзактов по цепочке блоков b, где транзакты подготавливаются блоком генератора g
`Block.runByStream(s, b)`	Возвращает действие, которое может быть применено к оператору do! для запуска обработки транзактов по цепочке блоков b, где транзакты берутся из потока событий s
`Block.runBySignal(s, b)`	Возвращает действие, которое может быть применено к оператору do! для запуска обработки транзактов по цепочке блоков b, где транзакты испускаются дискретным сигналом s

Здесь вторая функция является сокращенной записью для вызова первой функции с помощью выражения, которое создает генератор через `Block.run(GeneratorBlock.byStream(s), b)`. То же справедливо и для третьей функции, которая является сокращением для `Block.run(GeneratorBlock.bySignal(s), b)`.

Все три функции возвращают действие, которое еще должно быть выполнено с помощью оператора `do!`, что делает исполнение явным в уравнениях.

Следующий пример иллюстрирует простейшую законченную модель, но которая ничего не обрабатывает.

Пример

```
A = do! Block.runByStream(Stream.empty, Block.terminate);
```

Здесь пустой поток внешних событий обрабатывается завершающей цепочкой блоков. Ничего не происходит. Однако это показывает основную идею.

Ниже другой пример.

Пример

```
S = Stream.exponential(5);
B = Block.advance(3) >>> Block.terminate;
A = do! Block.runByStream(S, B);
```

Оператор do! может быть применен также к элементам массива.

Пример

```
S = [ Stream.exponential(i) | i <- 5..10 ];
B = [ Block.advance(i - 2) >>> Block.terminate | i <- 5..10 ];
A = [ do! Block.runByStream(S[i], B[i]) | i <- 5..10 ];
```

5.15 Очереди

В программе «ВизуальнаяАйвика» вводятся некоторые сущности для сбора статистики. Одной из таких сущностей является очередь. Транзакт може быть добавлен в очередь, а потом удален из нее. «ВизуальнаяАйвика» ведет подсчет всех этих операций.

Таблица 5.19: Конструктор очереди

`Queue.create()`	Создает новый объект очереди

Для того, чтобы помещать транзакты в очередь, а потом извлекать из нее, мы должны использовать соответствующие вычисления блоков. Эти вычисления имеют опциональные параметры приращения содержимого и его уменьшения, которые по умолчанию имеют значение, равное 1.

Таблица 5.20: Операции с очередью

`Block.queue(q)`	Возвращает блок, который помещает транзакты в очередь `q` с приращением содержимого, равным 1

`Block.queue(q, n)`	Возвращает блок, который помещает транзакты в очередь `q` с указанным приращением содержимого, равным `n`
`Block.depart(q)`	Возвращает блок, который извлекает транзакты из очереди `q` с уменьшением содержимого, равным 1
`Block.depart(q, n)`	Возвращает блок, который извлекает транзакты из очереди `q` с указанным уменьшением содержимого, равным `n`

Для примера мы можем поместить транзакты в очередь, задержать их, а затем извлечь из очереди. Обычно мы также могли бы попытаться заполучить некоторый ресурс сразу после помещения транзакта в очередь, но о ресурсах рассказывается далее.

Пример

```
Q = Queue.create();
B = Block.enqueue(Q) >>>
      Block.advance(12 + random(0, 10) + transact.ABC) >>>
      Block.depart(Q) >>> B2;
```

В любой момент модельного времени мы можем получить информацию о свойствах очереди, например для того, чтобы нарисовать их на графике, таком как график отклонения *Deviation chart*.

Таблица 5.21: Свойства очереди

`Queue.content(q)`	Возвращает значение содержимого очереди для текущего модельного времени

`Queue.contentStats(q)`	Возвращает сводную статистику по содержимому очереди на момент текущего модельного времени. Значение статистики привязано ко времени (статистика по времени)
`Queue.enqueueCount(q)`	Возвращает общее количество добавлений в очередь q для текущего модельного времени
`Queue.zeroEntry-EnqueueCount(q)`	Похоже на предыдущую функцию, но возвращает такое количество добавлений в очередь, где не было задержки между извлечением из очереди и помещением в нее
`Queue.waitTime(q)`	Возвращает сводную статистику по времени ожидания в очереди на момент текущего модельного времени. Значение статистики основано на наблюдениях (статистика по выборке)
`Queue.nonZeroEntry-WaitTime(q)`	Похоже на предыдущую функцию, но возвращает такую статистику, где была задержка между извлечением из очереди и помещением в нее

Например, мы создаем временную переменную `WaitTime` для сохранения статистики, которая затем может быть добавлена на график отклонения для отображения.

Пример

```
Q = Queue.create();
WaitTime = Queue.waitTime(Q);
```

5.16 Прибор

Другим видом сущностей, поддерживаемых в программе «ВизуальнаяАйвика», является прибор. Прибор — это такой ресурс, который может иметь только одного владельца. Прибор можно захватить, вытеснить, освободить и вернуть. При вытеснении ресурса прежний транзакт-владелец может быть перенаправлен на другую цепочку блоков для продолжения своего выполнения. Это позволяет нам моделировать достаточно сложное поведение.

Таблица 5.22: Конструктор прибора

`Facility.create()`	Создает новый объект прибора

Чтобы воздействовать на прибор транзактами, определены соответствующие вычисления блоков. Некоторые из этих вычислений могут иметь опциональные параметры. Наиболее сложное поведение присуще функции `Block.preempt`.

Таблица 5.23: Операции с приборами

`Block.seize(f)`	Возвращает блок, который пытается захватить прибор `f`. Позже этот прибор должен быть освобожден транзактом
`Block.release(f)`	Возвращает блок, который освобождает прибор `f`, который был ранее захвачен транзактом

`Block.preempt(f)`	Возвращает блок, который пытается вытеснить заданный прибор `f`. Позже этот прибор должен быть возвращен транзактом. См. ниже
`Block.preempt(f, priorityMode=f1, removalMode=f2)`	Возвращает блок, который пытается вытеснить заданный прибор `f`, где есть опциональные флаги `f1` и `f2` для режимов приоритета и удаления. По умолчанию оба булева флага выключены. См. ниже
`Block.preempt(f, priorityMode=f1, removalMode=f2, transfer(b) via transact.attribute)`	Подобно предыдущей функции, но если транзакт вытесняется, то он будет перенаправлен цепочке блоков `b`, гда заданный атрибут транзакта будет содержать значение оставшегося времени, которое транзакт еще должен был бы провести во время задержки. См. ниже
`Block.preempt(f, transfer(b) via transact.attribute)`	Более простая форма записи для предыдущей функции со значениям флагов режимов по умолчанию
`Block.return(f)`	Возвращает блок, который возвращает прибор `f`, который был ранее вытеснен транзактом

Наиболее популярным вариантом использования является подсчет статистики для очереди, когда мы патаемся захватить прибор, сымитировать некоторую активность с помощью задержки, а затем освободить или вернуть прибор обратно. Здесь транзакт блокируется в случае нехватки ресурса, что может привести к увеличению времени ожидания в очереди.

Пример

```
Q = Queue.create();
F = Facility.create();
B = Block.queue(Q) >>>
      Block.seize(F) >>>
      Block.depart(Q) >>>
      Block.advance(random(10, 20)) >>>
      Block.release(F) >>> B2;
```

Как это было справедливо и для очереди, мы можем запросить значения свойств и статистику для прибора во время имитации в любой момент модельного времени.

Таблица 5.24: Свойства прибора

`Facility.isInterrupted(f)`	Возвращает булев флаг, указывающий на то, прерван ли прибор `f` в текущем модельном времени
`Facility.count(f)`	Возвращает количество ресурса для прибора в текущем модельном времени
`Facility.countStats(f)`	Возвращает статистику по количеству ресурса для прибора на текущий момент модельного времени. Значение статистики зависит от времени (статистика по времени)
`Facility.captureCount(f)`	Возвращает общее количество захватов ресурса для прибора `f` на текущий момент модельного времени
`Facility.utilisationCount(f)`	Возвращает используемое количество ресурса для прибора `f` на текущий момент модельного времени

`Facility.utilisation-CountStats(f)`	Возвращает статистику по используемому количеству ресурса для прибора на текущий момент модельного времени. Значение статистики зависит от времени (статистика по времени)
`Facility.queueLength (f)`	Возвращает длину очереди для прибора `f` на текущий момент модельного времени
`Facility.queueLength-Stats(f)`	Возвращает статистику по длине очереди прибора на текущий момент модельного времени. Значение статистики зависит от времени (статистика по времени)
`Facility.totalWait-Time(f)`	Возвращает общее время ожидания для прибора `f` на текущий момент модельного времени
`Facility.waitTime(f)`	Возвращает статистику по времени ожидания в очереди прибора на текущий момент модельного времени. Значение статистики основано на наблюдениях (статистика по выборке)
`Facility.total-HoldingTime (f)`	Возвращает общее время удержания прибора `f` на текущий момент модельного времени
`Facility.holdingTime (f)`	Возвращает статистику по времени удержания прибора на текущий момент модельного времени. Значение статистики основано на наблюдениях (статистика по выборке)

Как и прежде, мы можем сохранить любое из этих свойств в некоторой переменной, а потом вывести результат.

Во время имитации мы можем запросить любое из этих свойств. Например, мы можем проверить, а не прерван ли прибор сейчас. Если прибор прерван, то мы можем завершить обработку транзакта, как показано в примере ниже.

Пример

```
Q = Queue.create();
F = Facility.create();
B = Block.select(if Facility.isInterrupted(F) then Busy else B2);
B2 = Block.preempt(F, priorityMode=true,
       transfer(Add) via transact.P5) >>> B3;
...
Busy = Block.terminate;
```

Также здесь вытесненный транзакт, прежний владелец прибора, будет перенаправлен в цепочку блоков Add, где атрибут транзакта с названием «P5» будет содержать значение оставшегося модельного времени, в течение которого транзакт еще должен был бы удерживаться во время задержки.

В разделе А из приложения к документу приведены технические детали того, как реализовано в симулятор вытеснение прибора.

5.17 Многоканальное устройство

Многоканальное устройство является другим видом ресурсов, поддерживаемых в программе «ВизуальнаяАйвика». Устройство имеет емкость. Также устройство может быть использовано многими транзактами, пока доступно содержимое.

Таблица 5.25: Конструктор устройства

`Storage.create(n)`	Создает новый объект устройства по заданной емкости `n`

Существуют вычисления блоков для использования устройства. Некоторые из этих вычислений могут иметь опциональные параметры.

Таблица 5.26: Операции с устройствами

`Block.enter(s)`	Возвращает блок, который пытается ввести транзакт в устройство `s` с уменьшением его содержимого на 1. Позже устройтво должно быть оставлено транзактом
`Block.enter(s, n)`	Возвращает блок, который пытается ввести транзакт в устройство `s` с уменьшением его содержимого на `n`. Позже устройтво должно быть оставлено транзактом
`Block.leave(s)`	Возвращает блок, который оставляет устройство `s` с увеличением его содержимого на 1. Транзакт должен был быть введен в устройство ранее
`Block.leave(s, n)`	Возвращает блок, который оставляет устройство `s` с увеличением его содержимого на `n`. Транзакт должен был быть введен в устройство ранее

При попытке войти в устройство, если содержимого недостаточно, то тогда транзакт блокируется. Однако это поведение много проще, чем захват или вытеснение прибора. Другим отличием является то, что содержимое может меняться не только на 1.

Обычным подходом будет подсчет статистики для очереди, когда мы патаемся ввести транзакт в устройство, сымитировать некоторую активность с помощью задержки, а затем вывести транзакт из устройства обратно. Здесь транзакт блокируется в случае нехватки ресурса, что может привести к увеличению времени ожидания в очереди.

Пример

```
Q = Queue.create();
```

```
S = Storage.create();
B = Block.queue(Q) >>>
      Block.enter(S) >>>
      Block.depart(Q) >>>
      Block.advance(random(10, 20)) >>>
      Block.leave(S) >>> B2;
```

У многоканального устройства есть свои свойства. Мы можем запросить их в любой момент модельного времени.

Таблица 5.27: Свойства устройства

`Storage.capacity(s)`	Возвращает емкость устройства
`Storage.content(s)`	Возвращает значение содержимого для устройства в текущий момент модельного времени
`Storage.content-Stats(s)`	Возвращает статистику по значениям содержимого устройства на текущий момент модельного времени. Значение статистики зависит от времени (статистика по времени)
`Storage.useCount(s)`	Возвращает общее количество случаев для устройства `s`, когда содержимое уменьшалось
`Storage.used-Content(s)`	Возвращает общее количество используемого содержимого для устройства `s` на текущий момент модельного времени
`Storage.content-Utilisation(s)`	Возвращает значение использования содержимого устройства для текущего модельного времени

`Storage.content-UtilisationStats(s)`	Возвращает статистику по значениям использования содержимого устройства на текущий момент модельного времени. Значение статистики зависит от времени (статистика по времени)
`Storage.queueLength(s)`	Возвращает длину очереди устройства для текущего модельного времени
`Storage.queueLength-Stats(s)`	Возвращает статистику по длине очереди устройства на текущий момент модельного времени. Значение статистики зависит от времени (статистика по времени)
`Storage.totalWait-Time(s)`	Возвращает общее время ожидания для устройства на текущий момент модельного времени
`Storage.waitTime(s)`	Возвращает статистику по времени ожидания для устройства на текущий момент модельного времени. Значение статистики основано на наблюдениях (статистика по выборке)
`Storage.average-HoldingTime(s)`	Возвращает среднее время удержания устройства на текущий момент модельного времени

Мы можем сохранить любое из этих свойств в некоторой переменной, а потом вывести результат.

5.18 Ансамбль транзактов

Каждый транзакт, созданный с помощью блока генератора, имеет свой собственный ансамбль транзактов. При этом, во время имитации транзакты могут быть разделены на множество копий, где каждая копия будет принадлежать тому же ансамблю транзактов. Тогда группа таких транзактов может быть скомпонована вместе в один транзакт, или они могут быть задержаны, пока все из них не соберутся вместе.

Также мы можем создать так называему цепочку сопряжения, так чтобы некоторый транзакт мог быть задержан до тех пор, пока другой транзакт из того же ансамбля транзактов не попытался бы синхронизироваться с заданной общей цепочкой сопряжения.

Таблица 5.28: Конструктор для цепочки сопряжения

`MatchChain.create()`	Создает новый объект цепочки сопряжения

Существуют следующие вычисления блоков, которые используют ансамбль транзактов, общий для целой группы транзактов, которые были разбиты ранее на копии из того же самого транзакта.

Таблица 5.29: Операции с ансамблем транзактов

`Block.split(n, b)`	Возвращает блок, который разделяет каждый транзакт на `n` копий, каждая из которых направляется в заданную цепочку блоков `b`. Каждая копия будет принадлежать одному и тому же ансамблю исходного транзакта
`Block.assemble(n)`	Возвращает блок, который объединяет `n` транзактов из одного и того же ансамбля в один транзакт при условии, что они были разделены на копии ранее

`Block.gather(n)`	Возвращает блок, который собирает вместе `n` транзактов из одного и того же ансамбля при условии, что они были разделены на копии ранее. После того, как `n` транзактов собраны вместе, их выполнение продолжается
`Block.matchChain(c)`	Возвращает блок, который задерживает транзакт, пока другой транзакт из того же самого ансамбля не вызовет похожий блок с той же самой цепочкой сопряжения `c`

Следующий пример показывает, как мы можем разбивать, а затем собирать транзакты вместе.

Пример

```
Worker = Facility.create();

S2 = Block.seize(Worker) >>>
       Block.advance(random(8 - 3, 8 + 3)) >>>
       Block.split(1, S2) >>>
       Block.release(Worker) >>>
       Block.priority(1) >>>
       Block.gather(24) >>> S3;
```

Здесь захватывается прибор, чтобы сделать копии транзакта с задержкой. Затем увеличивается приоритет, и собираются вместе 24 транзакта. На самом деле, в этом примере используется только один исходный транзакт, но в примере создается множество его копий.

5.19 Связующая цепочка

Связующая цепочка является разновидностью очереди, которая позволяет нам временно усыплять транзакты так, чтобы потом они могли быть разбужены (реактивированы) позже.

Таблица 5.30: Конструкторы для связующей цепочки

`LinkChain.createFIFO()`	Создает новый экземпляр связующей цепочки на основе очереди FIFO (первым вошёл - первым ушёл: First In, First Out)
`LinkChain.createLIFO()`	Создает новый экземпляр связующей цепочки на основе очереди LIFO (последним вошёл - первым ушёл: Last In, First Out)
`LinkChain.createSIRO()`	Создает новый экземпляр связующей цепочки на основе очереди SIRO (обслуживание в случайном порядке: Service In Random Order)
`LinkChain.createBy-Priorities()`	Создает новый экземпляр связующей цепочки на основе очереди приоритетов

Для использования связующих цепочек существуют соответствующие вычисления блоков. Некоторые из этих вычислений имеют особый синтаксис.

Таблица 5.31: Операции со связующими цепочками

`Block.link(c)`	Возвращает цепочку блоков, которая связывает текущий транзакт с заданной связующей цепочкой, а затем усыпляет транзакт, пока он не будет отсоединен позже

`Block.linkByPriority(c, p)`	Возвращает цепочку блоков, которая связывает текущий транзакт с заданной связующей цепочкой, `c`, по приоритету `p`, а затем усыпляет транзакт, пока он не будет отсоединен позже. Здесь `p` может быть выражением, которое может зависеть от атрибутов транзакта. В отличие от приоритетов транзактов меньшее значение `p` означает более высокий приоритет!
`Block.unlink(c, transfer(b))`	Возвращает блок, который отсоединяет верхний транзакт от заданной связующей цепочки `c`, а затем пробуждает тот транзакт, посылая его в заданную цепочку блоков `b`
`Block.unlink(c, transfer(b) if e)`	Возвращает блок, который отсоединяет транзакт от заданной связующей цепочки `c` по предикату `e`, а затем пробуждает тот транзакт, посылая его в заданную цепочку блоков `b`. Здесь `e` является логическим выражением, которое может зависеть от текущего транзакта по ключевому слову `transact`, а также зависеть от проверяемого транзакта по ключевому слову `that`

`Block.unlinkN(c, transfer(b), n)`	Возвращает блок, который отсоединяет заданное количество `n` верхних транзактов от связующей цепочки `c`, а затем пробуждает те транзакты, посылая их в заданную цепочку блоков `b`

Стоит заметить, как происходит процесс отсоединения при указании предиката. Процесс отсоединения от очередей FIFO и LIFO строго определен в соответствии с порядком сортировки. Что касается очереди приоритетов, то процесс разъединения, который происходит с помощью предиката, строго не определен, и это зависит от того, как двоичная куча содержит свои элементы, где двоичная куча используется для реализации очереди приоритетов.

Мы можем ссылаться на оба транзакта при указании предиката. Мы можем ссылаться с помощью ключевого слова `transact` на текущий транзакт, который вошел в блок. Также мы можем ссылаться с помощью ключевого слова `that` на транзакт, который проверяется и который содержится в очереди связующей цепочки.

Например, мы можем написать

Пример

```
B0 = Block.unlink(C1, transfer(B1)
       if transact.Time1 < that.Time1);
```

У связующей цепочки есть свои собственные свойства. Мы можем запросить их у связующей цепочки в любой момент модельного времени.

Таблица 5.32: Свойства связующей цепочки

`LinkChain.queue-Length(c)`	Возвращает длину очереди для связующей цепочки `c` на текущий момент модельного времени

`LinkChain.queue-LengthStats(c)`	Возвращает статистику по длине очереди связующей цепочки на текущий момент модельного времени. Значение статистики зависит от времени (статистика по времени)
`LinkChain.totalWait-Time(c)`	Возвращает общее время ожидания для связующей цепочки с на текущий момент модельного времени
`LinkChain.waitTime(c)`	Возвращает статистику по времени ожидания в очереди связующей цепочки на текущий момент модельного времени. Значение статистики основано на наблюдениях (статистика по выборке)
`LinkChain.average-ResidenceTime(c)`	Возвращает среднее время пребывания для связующей цепочки с на текущий момент модельного времени

Мы можем сохранить любое из этих свойств в некоторой переменной и затем отобразить результат.

Эти свойства также полезны, если мы желаем проверить, есть ли записи в очереди связующей цепочки, чтобы решить, куда нам следует перевести транзакт.

Пример

```
F1 = Facility.create();
C1 = LinkChain.createFIFO();

B1 = Block.select(if Facility.count(F1) > 0 then Cont1 else B2);
B2 = Block.select(if LinkChain.queueLength(C1) == 0
       then Cont1 else B3);
B3 = Block.link(C1);
```

```
Cont1 = Block.terminate;
```

Здесь мы проверяем условия, прежде чем принимаем решение о том, следует ли нам привязать транзакт к очереди.

Наконец, чтобы отсоединить все транзакты от связующей цепочки, мы можем запросить у связующей цепочки текущую длину очереди, а затем пробудить (реактивировать) указанное количество транзактов.

Пример

```
C1 = LinkChain.createFIFO();

B4 = Block.unlinkN(C1, transfer(Cont1),
       LinkChain.queueLength(C1));
```

5.20 Сигналы

Сигналом является некоторая активность, которая может возбуждать внешние события. Это отличается от потоков в том смысле, что потоки пассивны. Поток производит внешние события только тогда, когда события потока запрашиваются извне и друг за другом, все время. А вот сигнал является независимой активностью, которая сама решает, когда испускать внешние события. Сигнал активен.

Полезно то, что мы можем создавать блоки генератора и по потокам, и по сигналам. Это означает, что сигнал подобно потоку может инициировать обработку транзактов.

Таблица 5.33: Вычисления сигнала

`Signal.empty`	Возвращает пустой сигнал, который не испускает внешних событий
`Signal.merge(x, y)`	Объединяет два сигнала `x` и `y`

По определению, начальным значением для каждого сигнала является `Signal.empty`.

Здесь функция объединения создает новый сигнал, который испускается каждый раз, когда один из заданных сигналов испускается в

свою очередь. Это полезно, когда мы собираемся эффективно проверять условное выражение, как мы увидим далее.

При обработке транзакта соответствующее вычисление блока может приостанавливать свое выполнение в ожидании заданного сигнала. После прихода сигнала вычисление блока возобновляет свою обработку транзакта. Это — ключ для эффективной обработки условных выражений, потому что мы можем испускать события только в те моменты модельного времени, когда условное выражение должно быть перепроверено снова.

Таблица 5.34: Ожидание сигналов

`Block.await(s)`	Возвращает блок, ожидающий отправки указанного сигнала при обработке транзакта
`Block.await (transact.attribute <- s)`	Возвращает блок, ожидающий отправки указанного сигнала при обработке транзакта. Все значения сигнала заполняются в транзакте по указанному префиксу атрибута

Вторая функция позволяет нам ввести значения сигнала в контекст текущего транзакта.

Например, если у сигнала был атрибут `Value`, и мы указали вычисление `Block.await(transact.Sig <- s)`, то значение сигнала будет помещено во вложенный атрибут транзакта `Sig.Value`. Затем мы можем запросить соответствующее значение, используя выражение `transact.Sig.Value`.

Причина в использовании вложенных атрибутов кроется в том, что сигнал подобно транзакту может иметь несколько атрибутов.

Сигналы могут быть созданы с помощью изменяемых ссылок, которые рассматриваются в разделе 5.22.

Наконец, мы можем создать пуассоновский сигнал по заданной показательно распределенной средней задержке, которая может меняться во времени. Это открывает двери на пути к гибридному моделированию.

Таблица 5.35: Пуассоновский сигнал

`Signal.exponential(m)`	Возвращает сигнал о приходе событий, для которых случайные задержки распределены экспоненциально со средним `m`, которое может меняться во времени, например, которое может зависеть от значений интегралов

По такому сигналу мы можем начать обработку транзактов. Есть и другая возможность. Мы можем оценить интенсивность транзактов, где такая оценка может быть использована в дифференциальных уравнениях. Пожайлуста, посмотрите главу 9 для дополнительной информации.

5.21 Статистика

Сбор и обработка статистики — важная часть имитации. Программа «ВизуальнаяАйвика» использует подход, где сводная статистика трактуется как неизменяемая структура данных, что упрощает моделирование и делает имитацию более надежной.

Существует два разных типа статистики, которую мы можем собирать. Первый тип основан на наблюдениях, тогда как второй основан на выборке, зависящей от модельного времени.

5.21.1 Статистика на основе наблюдений

Первый тип данных используется для аккумулирования статистики, основанной на наблюдениях. Примером является время ожидания в очереди.

Таблица 5.36: Создание статистики на основе наблюдений

`SamplingStats.emptyInts`	Возвращает пустую статистику по наблюдениям для целых чисел
`SamplingStats.emptyFloats`	Возвращает пустую статистику по наблюдениям для вещественных чисел
`SamplingStats.add(v, st)`	Добавляет указанное наблюдение `v` в статистику `st`, создавая другой экземпляр статистики
`SamplingStats.append(st1, st2)`	Объединяет оба значения статистики, создавая новый экземпляр статистики

Важно заметить, что экземпляры `SamplingStats.emptyInts` и `SamplingStats.emptyFloats` возвращают разные и даже несовместимые значения статистики, которые не могут быть объединены вместе с помощью функции `SamplingStats.append`.

Статистика может быть отображена в некоторых элементах для вывода результатов, таких как график отклонения, но мы также можем запросить у статистики некоторые свойства, которые могут быть либо отображены, либо использованы в рамках моделирования.

Таблица 5.37: Свойства статистики на основе наблюдений

`SamplingStats.count(st)`	Возвращает общее количество наблюдений в выборке

`SamplingStats.min(st)`	Возвращает минимальное значение для выборки
`SamplingStats.max(st)`	Возвращает максимальное значение для выборки
`SamplingStats.mean(st)`	Возвращает среднее значение
`SamplingStats.mean2(st)`	Возвращает среднее квадратическое значение
`SamplingStats.variance(st)`	Возвращает оценку дисперсии
`SamplingStats.deviation (st)`	Возвращает среднеквадратическое отклонение

5.21.2 Статистика с привязкой ко времени

Второй тип статистики очень похож, но только он уже позволяет использовать выборку, привязанную к точкам модельного времени. Соответствующую случайную величину на английском языке часто называют *time-persistent*[2]. Примером такой случайной величины является длина очереди.

На русском языке для обозначения самой статистики используется иногда термин *непрерывной статистики*[4]. Для краткости мы будет использовать термин *временной статистики*, поскольку он ближе к названию соответствующего типа данных в самой программе «ВизуальнаяАйвика».

Таблица 5.38: Создание временной статистики

`TimingStats.emptyInts`	Возвращает пустую временную статистику для целых чисел

`TimingStats.emptyFloats`	Возвращает пустую временную статистику для вещественных чисел
`TimingStats.add(t, v, st)`	Добавляет для времени `t` указанное наблюдение `v` в статистику `st`, создавая другой экземпляр статистики

Статистика может быть отображена в некоторых элементах для вывода результатов, таких как график отклонения, но мы также можем запросить у статистики некоторые свойства, которые могут быть либо отображены, либо использованы в рамках моделирования.

Таблица 5.39: Свойства значения временной статистики

`TimingStats.count(st)`	Возвращает общее количество наблюдений в выборке
`TimingStats.min(st)`	Возвращает минимальное значение для выборки
`TimingStats.max(st)`	Возвращает максимальное значение для выборки
`TimingStats.last(st)`	Возвращает последнее значение для выборки
`TimingStats.minTime(st)`	Возвращает время моделирования, в которое был достигнут минимум
`TimingStats.maxTime(st)`	Возвращает время моделирования, в которое был достигнут максимум

`TimingStats.startTime(st)`	Возвращает время начальной выборки
`TimingStats.lastTime(st)`	Возвращает время последней выборки
`TimingStats.sum(st)`	Возвращает взвешенную сумму значений
`TimingStats.sum2(st)`	Возвращает взвешенную сумму квадратов значений
`TimingStats.mean(st)`	Возвращает среднее значение
`TimingStats.mean2(st)`	Возвращает среднее квадратическое значение
`TimingStats.variance(st)`	Возвращает оценку дисперсии
`TimingStats.deviation(st)`	Возвращает среднеквадратическое отклонение

5.22 Изменяемая ссылка

Изменяемые ссылки являются важной частью дискретно-событийных имитационных моделей. В программе «ВизуальнаяАйвика» мы можем использовать ссылки для хранения значений следующих типов: целые числа, числа с плавающей запятой, основанная на наблюдениях статистика и временная статистика. Как только ссылка получит свое начальное значение, тип данных ссылки не может быть изменен. Все последующие значения должны иметь тот же тип данных.

Таблица 5.40: Операции с изменяемой ссылкой

`Ref.create(v)`	Создает новую изменяемую ссылку по заданному начальному значению
`ref(r)`	Возвращает значение ссылки

`Ref.changed(r)`	Возвращает сигнал, который срабатывает каждый раз, когда ссылке присваивается значение

Что касается последней функции, то стоит заметить, что соответствующий сигнал будет испускать события с атрибутом `Value`, которому будет присвоено текущее значение ссылки.

Мы можем обновлять значения ссылки в рамках вычислений блоков, используя следующий синтаксис.

Таблица 5.41: Обновление изменяемой ссылки

`Block.assign(ref(r) <- v)`	Возвращает блок, который назначает заданной ссылке `r` значение `v` при обработке транзакта

Изменяемые ссылки полезны для хранения статистики. Например, мы можем указать время начала обработки транзакта в каком-либо атрибуте транзакта. Затем мы определяем фактическое время обработки и добавляем его к статистике, хранящейся в изменяемой ссылке.

Пример

```
Tatym = Ref.create(SamplingStats.emptyFloats);

S1 = Block.assign(transact.ArrivalTime <- time) >>> S2;

...

S13 = Block.assign(ref(Tatym) <-
  SamplingStats.add(time - transact.ArrivalTime,
    ref(Tatym))) >>> S14;

S14 = Block.terminate;
```

5.22.1 Эффективная проверка условных выражений

Иногда нам нужно протестировать условное выражение в цикле до тех пор, пока условие не будет выполнено. Мы можем эффективно это

сделать с помощью ссылок и сигналов.

Допустим, что мы имеем две ссылки R1 и R2. Нам нужно продолжить обработку транзакта только после того, как сумма значений этих двух ссылок станет больше, чем 5.

Решение следующее.

Пример

```
R1 = Ref.create(0);
R2 = Ref.create(0);

S = Signal.merge(Ref.changed(R1), Ref.changed(R2));

B1 = Block.select(if ref(R1) + ref(R2) > 5 then N1 else B2);
B2 = Block.await(S) >>> B3;
B3 = Block.transfer(B1);

N1 = Block.terminate;
```

Сначала мы проверяем условие в вычислении блока B1. Если условие выполнено, то мы выходим из цикла.

Сигнал S испускается каждый раз, как только сумма может измениться. Если условие было ложным, то тогда мы ждем сигнала. Здесь компьютер не потребляет процессорного времени! Он именно, что ждет момента, в который одна из ссылок изменится.

После того, как сигнал будет испущен, мы повторяем цикл, используя уровень косвенности в B3, чтобы разрешить циклическую зависимость.

N1 просто обозначает выход из цикла. Это могло бы быть произвольным вычислением цепочки блоков.

5.22.2 Особенности модельного времени

Есть тонкие места для понимания касательно того, как модельное время обрабатывается в дифференциальных уравнениях и дискретно-событийной имитации.

Допустим, что у нас есть ссылка R, которую мы хотим наблюдать в переменной X.

Пример

```
R = Ref.create(0);
X = ref(R);
```

Особенность заключается в том, что переменная X обновляется только во временных точках интегрирования. На самом деле, переменная R никогда не будет обновляться в точности в таких точках интегрирования, но мы будем брать ее снимок для обновления X.

При отображении графиков мы увидим именно снимки переменных во временных точках интегрирования.

Другая важная особенность связана с тем, что очередь событий использует нестабильную сортировку событий. Буквально это означает, что если два события имеют одинаковый приоритет и одинаковое время активации, то тогда порядок активации этих обоих событий, вообще говоря, непредсказуем. Это было сделано для эффективной сортировки событий. Однако, если приоритет или время активации отличаются, то тогда порядок активации событий строго предсказуем.

5.23 Функции конвейера

Конвейер похож на интеграл, но в первом используются значения интенсивности вместо производных. Также конвейер имитирует сам процесс передачи входных данных с помощью внутреннего списка, по которому передаются данные. Но в остальном же конвейер похож на интеграл. Притоки подобны производным. Значение конвейера подобно сумме интеграла.

В настоящее время конвейеры доступны только в версии программы «ВизуальнаяАйвика», которая идет с компилятором уравнений.

Таблица 5.42: Конструкторы конвейера

`conveyor(i, tt)`	Возвращает дискретный конвейер с входными данными интенсивности `i`, которые имеют время прохождения `tt`

`conveyor(i, tt, d)`	Возвращает дискретный конвейер с входными данными интенсивности `i`, которые имеют время прохождения `tt`. Аргумент `d` определяет скорость убывания
`conveyor(i, tt, d, c)`	Возвращает дискретный конвейер с входными данными интенсивности `i`, которые имеют время прохождения `tt`. Аргумент `d` определяет скорость убывания. Конвейер имеет емкость `c`
`conveyor(i, tt, d, c, iv)`	Возвращает дискретный конвейер с входными данными интенсивности `i`, которые имеют время прохождения `tt`. Аргумент `d` определяет скорость убывания. Конвейер имеет емкость `c`. Аргумент `iv` задает начальное значение

После того, как конвейер определен, мы можем запросить его значение. Также мы можем запросить некоторые его свойства, которые являются потоками. Каждый выходной поток возвращает значение интенсивности. Это также похоже на значение производной.

Таблица 5.43: Свойства конвейера

`outflow(s)`	Возвращает интенсивность оттока для конвейера `s`
`overflow(s)`	Возвращает интенсивность переполнения для конвейера `s`
`leakflow(s)`	Возвращает интенсивность тока утечки для конвейера `s`

Часто бывает проще создавать конвейеры с помощью «Диаграммы потоков и накопителей», используя редактор диаграмм. Но сам

редактор использует именно эти упомянутые функции.

Конвейеры могут быть размещены в массивах.

Если мы определяем время прохождения, то мы определяем, как входные данные становятся исходящим оттоком. Если мы определяем емкость, то может произойти переполнение, когда внутренний объем превышает указанное значение емкости. Если мы определяем ненулевую скорость убывания, то мы фактически определяем поток утечки. Все потоки возвращают значения интенсивности, которые по своей природе аналогичны производным.

5.24 Счетчик транзактов

Для возведения моста между дифференциальными уравнениями и дискретно-событийной моделью нам нужно уметь оценивать интенсивность транзактов. Соответствующий тип данных называется просто *счетчиком транзактов* (*англ.* the transact counter).

Таблица 5.44: Конструктор для счетчика транзактов

`TransactCounter.create()`	Создает новый счетчик транзактов

После того, как счетчик транзактов создан, мы можем регистрировать текущие транзакты в рамках вычислений блоков.

Таблица 5.45: Блок счетика транзактов

`Block.count(c)`	Возвращает блок, который регистрирует текущий транзакт заданным счетчиком транзактов `c`

После того, как транзакты зарегистрированы, мы можем оценить их интенсивность для использования в дифференциальных уравнениях. Есть и другие счетчики.

Таблица 5.46: Свойства счетчика транзактов

`TransactCounter.rate(c)`	Возвращает текущую интенсивность транзактов, зарегистрированных с помощью заданного счетчика транзактов `c`, где используется интервал времени `dt`
`TransactCounter.count(c)`	Возвращает текущее количество транзактов, зарегистрированных с помощью заданного счетчика транзактов `c`, где используется интервал времени `dt`
`TransactCounter.total-Count(c)`	Возвращает общее количество транзактов, зарегистрированных с помощью заданного счетчика транзактов `c`

Пожалуйста, посмотрите главу 9 для примеров того, как счетчики транзактов могут быть использованы в гибридном моделировании.

5.25 Строки

Строковые литералы должны быть заключены в кавычки следующим образом:

Пример

```
FileName1 = "/home/david/file-1.csv";
FileName2 = "/home/david/file-\"1 2 3\".csv";
```

Строки могут быть полезны, например, если мы собираемся передавать имена файлов в параметрах внешним модулям.

5.26 Вложенные модули

Модель может содержать модули, которые могут быть вложенными. Каждый модуль определяет свое собственное пространство имен для переменных.

«ВизуальнаяАйвика» использует модули для хранения переменных из одной и той же диаграммы в отдельном модуле.

Пример

```
// глобальная переменная
A = 1;

module M1 {

    // переменная M1.B
    B = A + 1;

    module M2 {

        // переменная M1.M2.C
        C = 2 * B;

        // использовать полное название для M1.B
        D = C - M1.B;
    }
}
```

Глава 6

Вывод результатов

Элемент для вывода результатов с панели инструментов диаграммы позволяет вам видеть результаты моделирования предпочтительным способом. Рисунок 6.1 показывает соответствующий элемент на панели инструментов. Тогда представление выводимого результата может быть графиком, или может быть таблицей с данными CSV, или чем-то другим. Элемент задает в точности то, как результаты будут представлены. Диаграмма может содержать произвольное количество элементов с результатами.

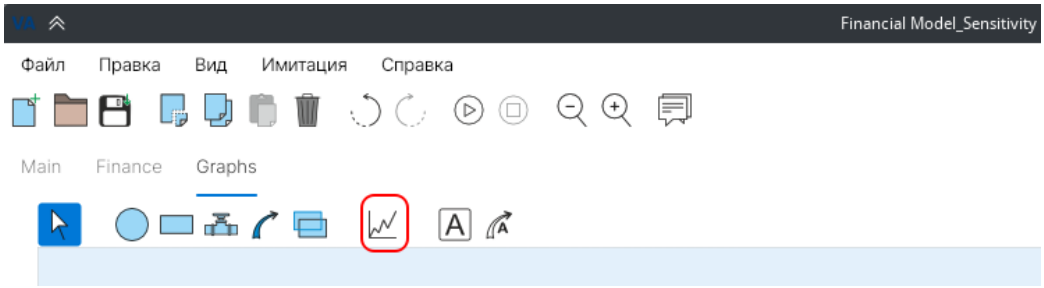


Рис. 6.1: Элемент для вывода результатов на панели диаграммы.

6.1 График отклонения

График отклонения (*англ.* Deviation Chart) — это наиболее универсальный способ представления, который применим как к единичному

запуску, так и ко множеству запусков. Если запуск единичный, то тогда график отклонения становится похожим на график временного ряда из раздела 6.3. Однако если используется вычислительный эксперимент по методу Монте-Карло, то тогда график отклонения показывает тренды и доверительные интервалы по правилу 3-х сигм.

Рисунок 6.2 иллюстрирует, как мы можем выбрать соответствующее представление для графика отклонения в редакторе вывода результатов. Этот редактор сразу открывается, как только вы выберете на диаграмме какой-либо элемент для отображения результатов. Здесь вам следует задать значение поля *Тип результата* равным *Deviation Chart*, а затем нажать на кнопку *Применить*, которая не показана на рисунке.

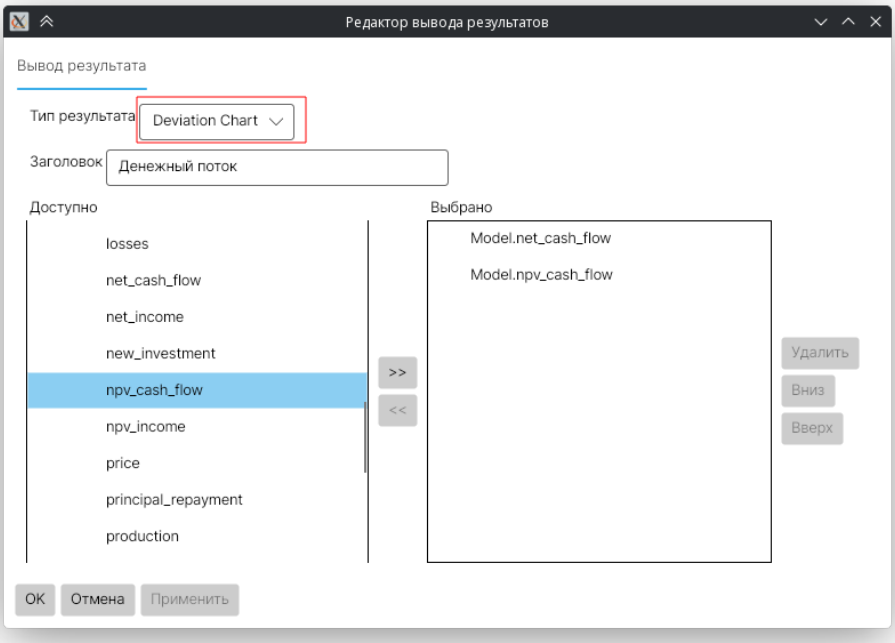


Рис. 6.2: Выбор представления для графика отклонения.

Рисунок 6.3 показывает, как график отклонения может выглядеть на диаграмме после окончания имитации.

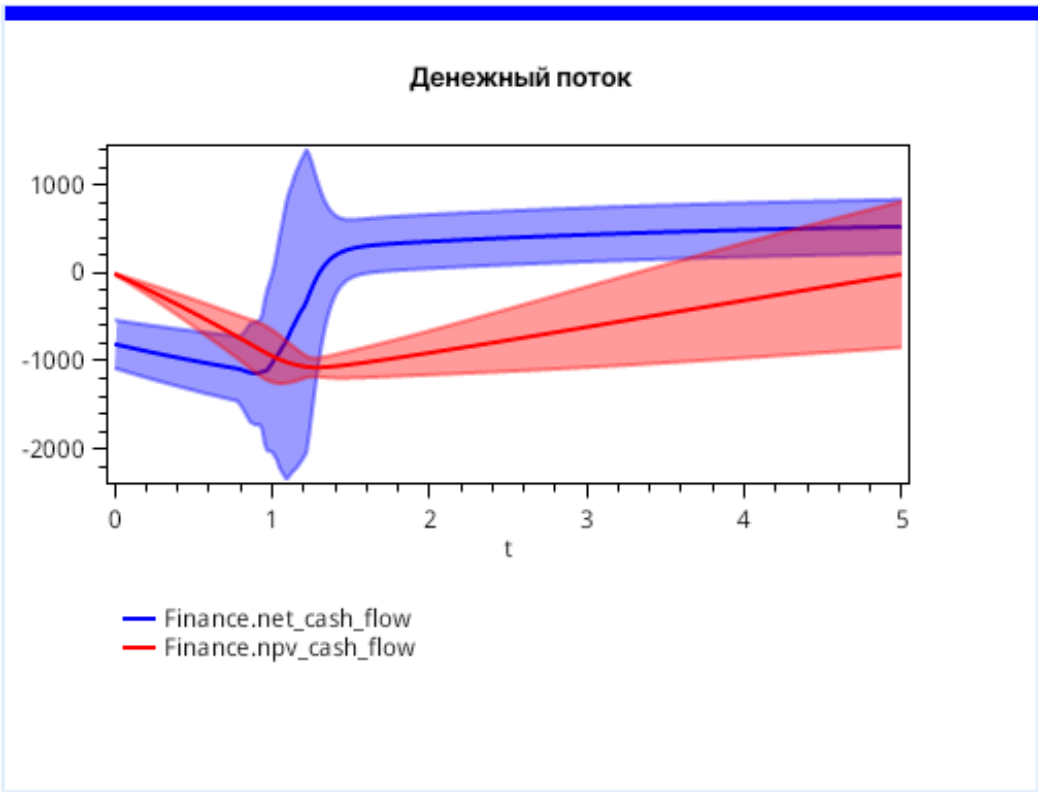


Рис. 6.3: Пример графика отклонения.

Представление графика отклонения может также показывать статические данные. Например, оно может отобразить время ожидания в очереди или статистику по содержимому прибора. Здесь мы предполагаем, что значения статистики распределены одинаково для разных имитационных запусков, но при этом сами запуски имитации не зависят друг от друга.

6.2 График экстремумов

График экстремумов очень похож на график отклонения. Только он показывает тренд, минимальное и максимальное значения.

Есть разница, если отображается статистика или обычная переменная. Переменная со статистикой всегда показывает исторические

значения минимума и максимума, тогда как экстремумы для обычной переменной вычисляются только в текущей временной точке интегрирования для всех имитационных запусков при использовании метода Монте-Карло.

Подобно показанному на рисунке 6.2, для отображения графика экстремумов необходимо выбрать для поля *Тип результата* значение *Extremum Chart*.

Рисунок 6.4 показывает, как график экстремумов может выглядеть на диаграмме для множества имитационных запусков.

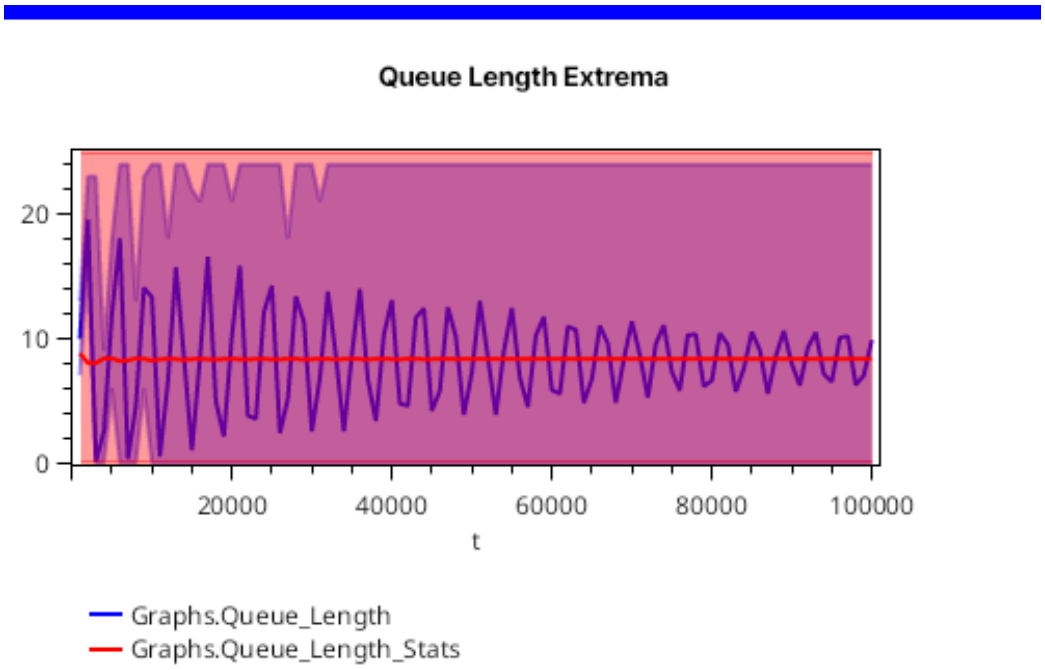


Рис. 6.4: Пример графика экстремумов.

6.3 Временной ряд

График временного ряда (*англ.* Time Series) отображает обычный график, где временной ряд зависит от модельного времени. Если используется метод Монте-Карло, то тогда будет нарисовано соответствующее количество графиков, по одному на каждый запуск. Поэтому следует использовать представление временного ряда только для единичного запуска или для метода Монте-Карло с малым количеством запусков.

Подобно показанному на рисунке 6.2, для отображения временного ряда необходимо выбрать для поля *Тип результата* значение *Time Series*, которое установлено по умолчанию.

Рисунок 6.5 показывает, как график временного ряда может выглядеть на диаграмме для единственного имитационного запуска.

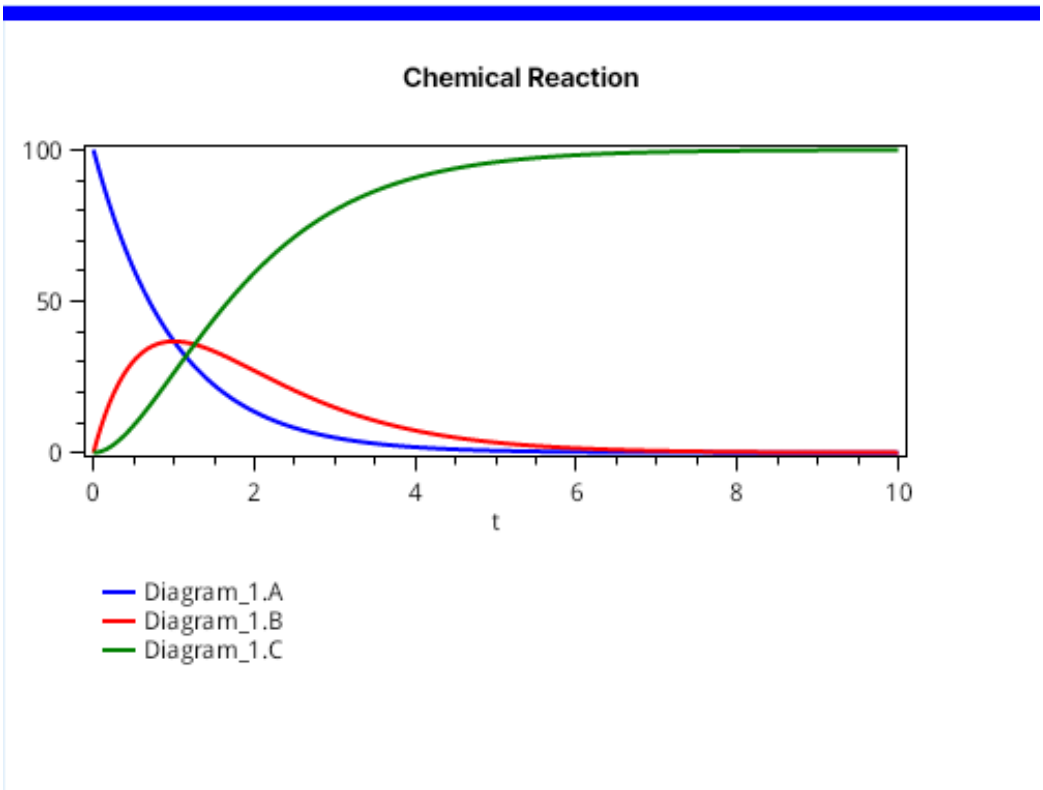


Рис. 6.5: Пример графика временного ряда.

6.4 График XY

График XY (*англ.* XY Chart) похож на график временного ряда, описанный в предыдущем разделе 6.3. Только первый выбранный ряд становится координатой X для графика. Как и прежде, если используется метод Монте-Карло, то тогда будет нарисовано соответствующее количество графиков, по одному на каждый запуск. Поэтому следует использовать представление графика XY только для единичного запуска или для метода Монте-Карло с малым количеством запусков.

Подобно показанному на рисунке 6.2, для отображения графика XY необходимо выбрать для поля *Тип результата* значение *XY Chart*.

Рисунок 6.6 показывает, как график XY может выглядеть на диаграмме.

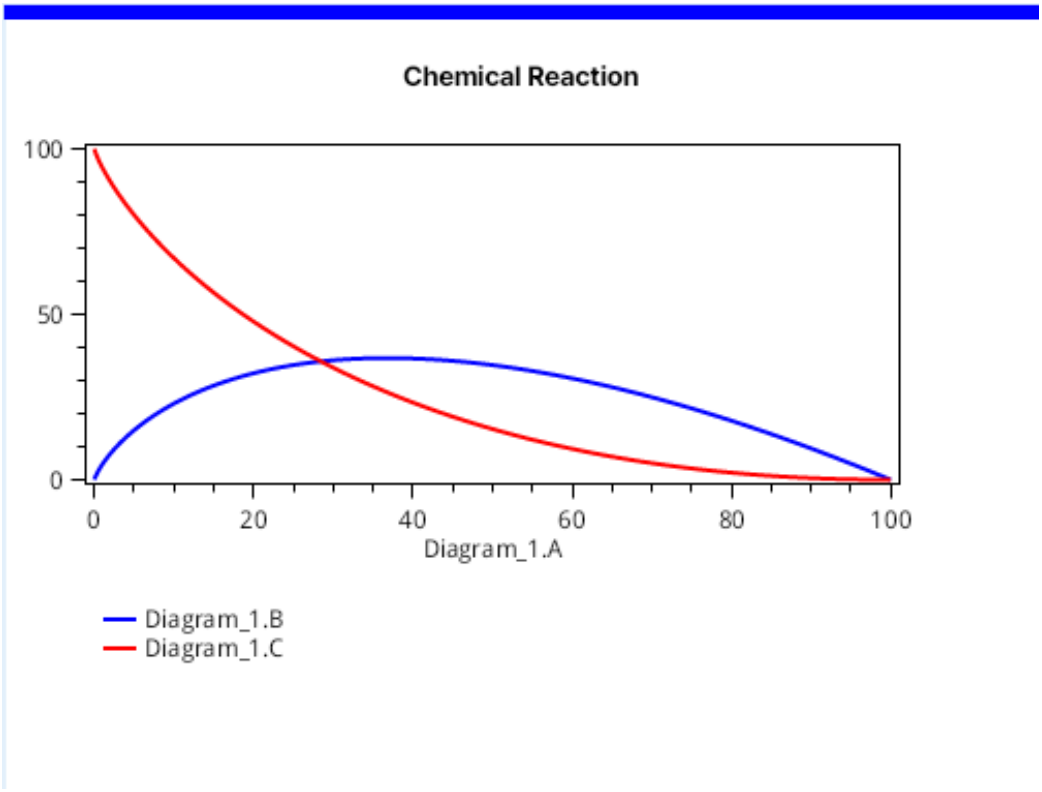


Рис. 6.6: Пример графика XY.

6.5 Таблица CSV

Таблица CSV (*англ.* CSV Table) предназначена для экспорта данных CSV в другие приложения или для сохранения данных в файл. Отображается компонент текстового блока, откуда вы можете скопировать соответствующие данные CSV. Если используется метод Монте-Карло, то тогда будет создано соответствующее количество компонентов, по одному на каждый запуск. Поэтому следует использовать представление таблицы CSV только для единичного запуска или для метода Монте-Карло с малым количеством запусков.

Подобно показанному на рисунке 6.2, для отображения таблицы CSV необходимо выбрать для поля *Тип результата* значение *Table*.

Рисунок 6.7 показывает, как таблица CSV может выглядеть на диаграмме.

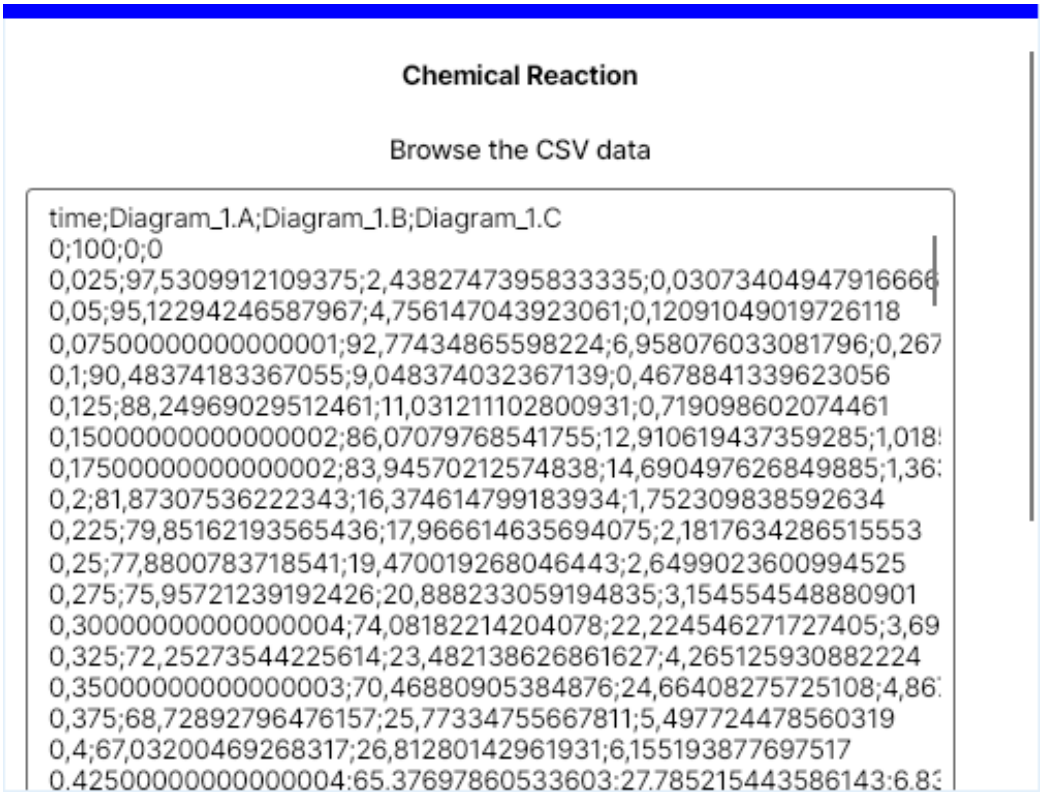


Рис. 6.7: Пример таблицы CSV.

6.6 Последние значения

Представление последних значений (*англ.* Last Values) предназначено для отображения значений рядов в конечной точке имитации. Если используется метод Монте-Карло, то тогда будет создано соответствующее количество компонентов, по одному на каждый запуск. Поэтому следует использовать представление последних значений только для единичного запуска или для метода Монте-Карло с малым количеством запусков.

Подобно показанному на рисунке 6.2, для отображения последних значений переменных необходимо выбрать для поля *Тип результата* значение *Last Values*.

Рисунок 6.8 показывает, как соответствующий элемент может выглядеть на диаграмме.

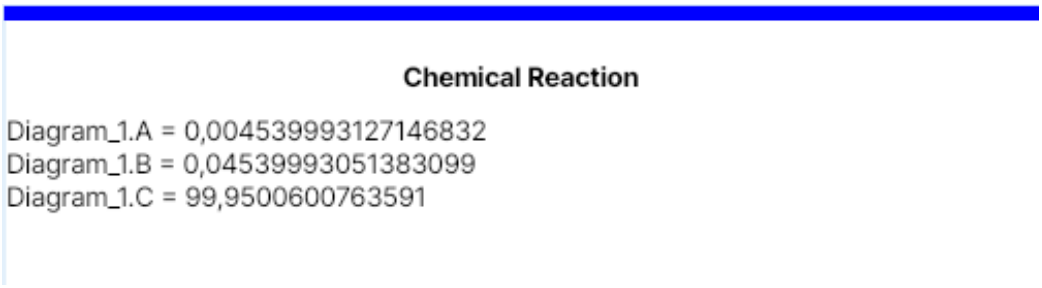


Рис. 6.8: Пример представления последних значений.

При отображении статистических данных этот элемент также показывает минимальное и максимальное значения, также как среднее и отклонение. Более того, этот элемент способен отображать сразу все свойства вместе для очереди, прибора и многоканального устройства.

6.7 Гистограмма последних значений

Представление гистограммы последних значений (*англ.* Last Value Histogram) предназначено для отображения гистограммы по значениям рядов в конечной модельной точке при использовании метода Монте-Карло со множеством запусков.

Подобно показанному на рисунке 6.2, для отображения гистограммы последних значений переменных необходимо выбрать для поля *Тип результата* значение *Last Value Histogram*.

Рисунок 6.9 показывает, как соответствующий элемент может выглядеть на диаграмме.

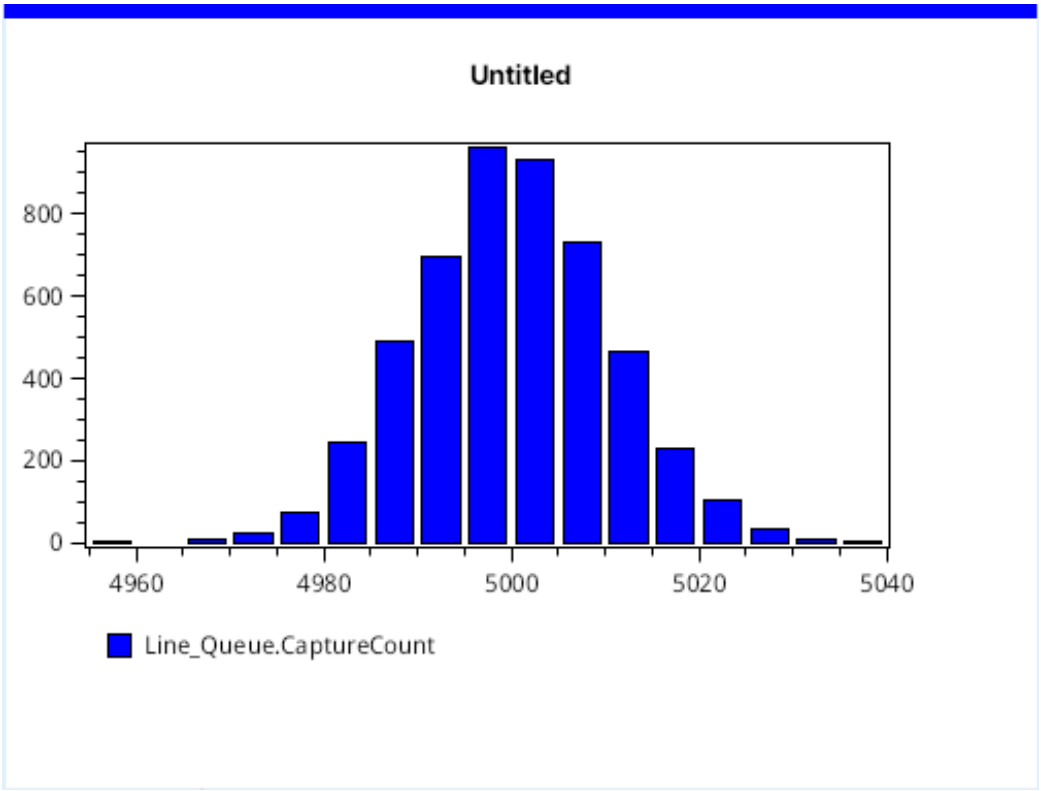


Рис. 6.9: Пример гистограммы последних значений.

6.8 Таблица CSV последних значений

При использовании метода Монте-Карло представление таблицы CSV последних значений (*англ.* Last Value CSV Table) собирает данные в конечных точках моделирования. Это представление предназначено для экспорта данных в другие приложения или для записи данных в

файл. Отображается компонент текстового блока, откуда вы можете скопировать соответствующие данные CSV.

Подобно показанному на рисунке 6.2, для отображения таблицы CSV последних значений переменных необходимо выбрать для поля *Тип результата* значение *Last Value Table*.

Рисунок 6.10 показывает, как соответствующий элемент может выглядеть на диаграмме.

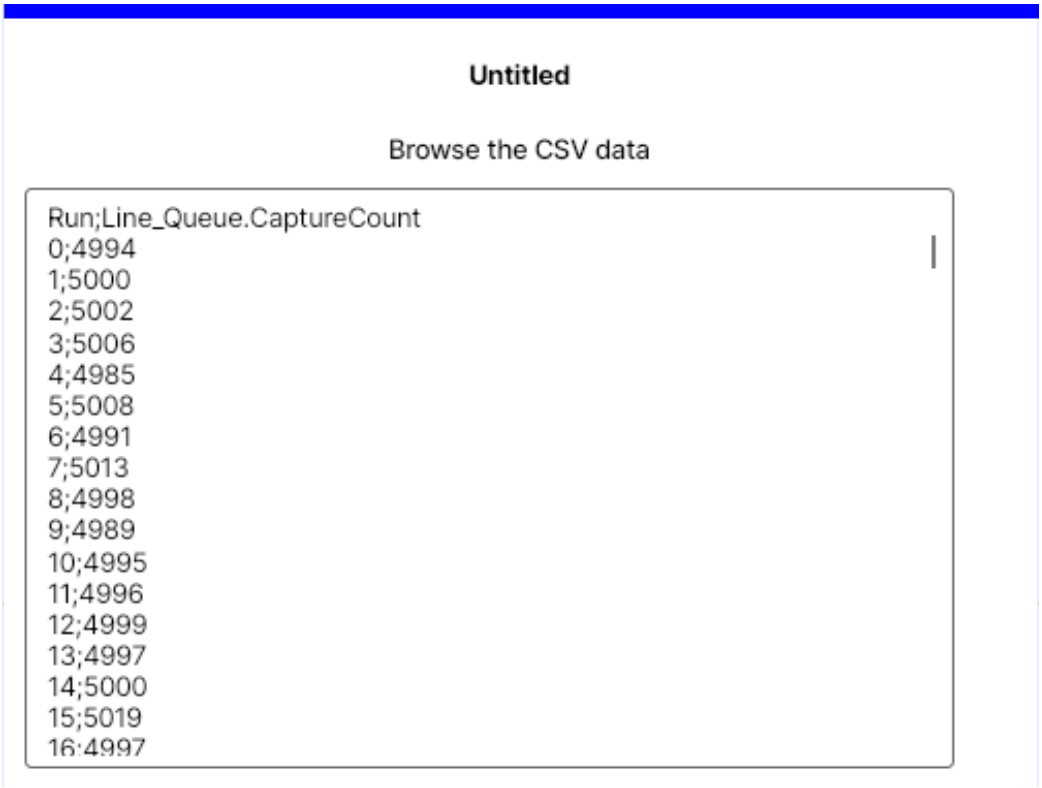


Рис. 6.10: Пример таблицы CSV последних значений.

6.9 Сводная статистика по последним значениям

При использовании метода Монте-Карло представление сводной статистики по последним значениям (*англ.* Last Value Statistics Summary)

собирает данные в конечных точках моделирования, а затем представляет результаты на соответствующем компоненте.

Подобно показанному на рисунке 6.2, для отображения статистики по последним значениям переменных необходимо выбрать для поля *Тип результата* значение *Last Value Statistics*.

Рисунок 6.11 показывает, как соответствующий элемент может выглядеть на диаграмме.

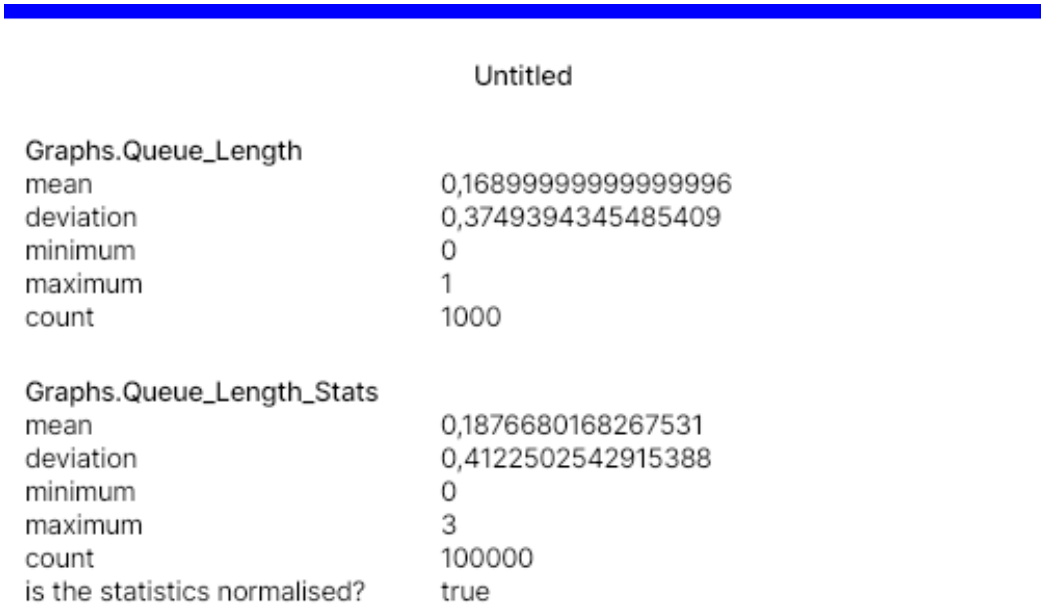


Рис. 6.11: Пример сводной статистики последних значений.

Есть один тонкий момент, связанный с отображением параметра `count` для статистики, зависящей от времени. Перед своим отображением временная статистика преобразуется в похожее представление, но на основе наблюдений, где параметр `count` начинает показывать совершенно другое значение, тогда как остальные параметры остаются корректными. В программе «ВизуальнаяАйвика» такая преобразованная статистика называется *нормализованной*. Так что, этот элемент

не показывает параметр `count` для временной статистики. В остальных случаях параметр `count` является тем, чем он и предполагается быть.

Например, на упомянутом рисунке 6.11 изображены два объекта статистической сводки. Первый объект — это обычная статистика на основе наблюдений. Второй же блок соответствует преобразованной версии временной статистики, где параметр `count` становится результатом нормализации.

Здесь важно отметить, что поправка Бесселя не применяется к нормализованной версии статистики.

Глава 7

Потоки и накопители

Потоки и накопители находятся в самом сердце системной динамики. Они связаны с обыкновенными дифференциальными уравнениями, но в действительности они могут представлять собой больше сущностей, чем только интегралы. Накопитель может быть интегралом или конвейером, где последний в чем-то похож на интеграл. Потоки являются значениями производных для интегралов. Однако если мы используем конвейер, то тогда потоки представляют собой уже соответствующие значения интенсивности, которые также показывают, как значение конвейера меняется с течением модельного времени.

Таким образом, накопитель — это некоторая абстракция для некоторой суммы элементов. Поток — это абстракция того, как такая сумма меняется.

7.1 Накопитель интеграла

На самом деле, интегралы могут быть созданы с помощью функции `integ`. Тем не менее, интеграл может быть также представлен на диаграмме с помощью элемента накопителя, где значения производных становятся элементами потока. Значение двунаправленного потока может иметь любой знак, тогда как значение однонаправленного потока всегда неотрицательно. Положительный приток (положительный входящий поток) увеличивает значение производной, тогда как положительный отток (положительный исходящий поток) уменьшает. Если знак потока отрицательный, то тогда эффект противоположен.

Диаграмма визуально показывает, представляет ли значение производной собой приток или отток. Она также показывает, является ли значение потока однонаправленным. Соответствующий элемент потока показывает направление. Если направление идет в сторону элемента накопителя, то это — приток. Если направление идет от элемента накопителя, то это — отток. Двунаправленный поток имеет две стрелки на своих концах, тогда как однонаправленный поток имеет только одну стрелку. В любом случае, вы всегда можете открыть вкладку *Уравнения* для того, чтобы проверить, что диаграмма корректна.

Здесь мы возвращаемся к модели из главы 2. Соответствующая диаграмма показана на рисунке 7.1.

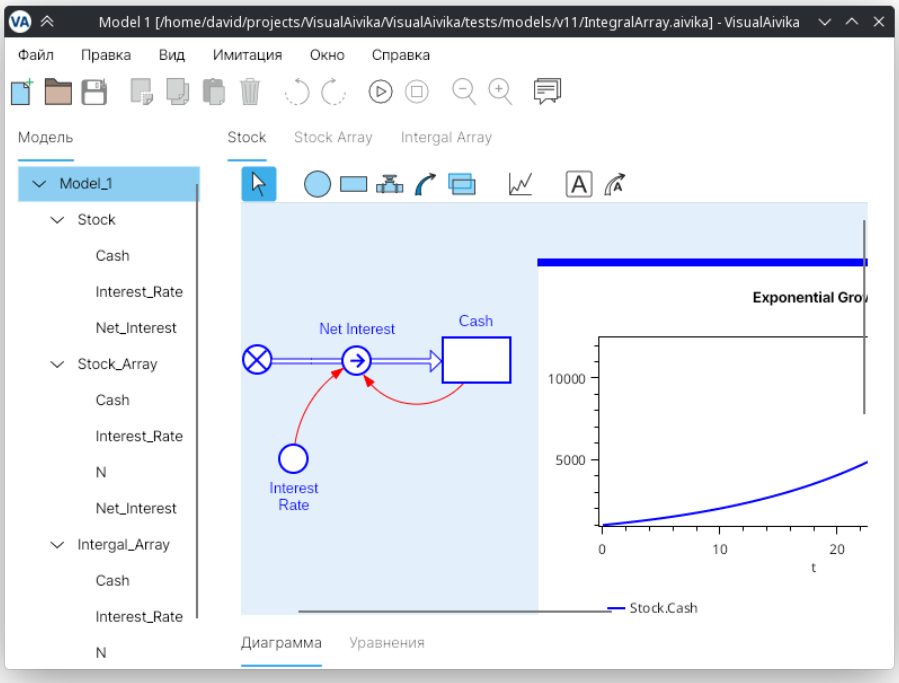


Рис. 7.1: Накопитель интеграла на диаграмме.

Мы можем видеть, что переменная `Net Interest` обозначает однонаправленный приток для накопителя `Cash`. Последний на самом

деле является интегралом, как мы можем увидеть по соответствующим уравнениям.

Пример

```
module Stock {

  Cash = integ (Net_Interest, 1000);
  Interest_Rate = 0.07;
  Net_Interest = max (0, Cash * Interest_Rate);

}  // Stock
```

7.2 Массив интегралов

Чтобы создать массив интегралов, мы могли в действительности переписать наши уравнения так, чтобы интегралы были представлены через так называемый дополнительный элемент вместо накопителя. Общий синтаксис массивов достаточно гибкий. Мы можем использовать этот синтаксис для определения массива интегралов.

Рисунок 7.2 отображает соответствующую диаграмму.

Соответствующие уравнения могут быть следующими. Они используют общий и довольно-таки гибкий синтаксис массивов.

Пример

```
module Intergal_Array {

  Cash = [ integ(Net_Interest[i], 1000) | i <- 1..N ];
  Interest_Rate = [ 0.07 + i * 0.01 | i <- 1..N ];
  N = 3;
  Net_Interest =
    [ max(0, Cash[i] * Interest_Rate[i]) | i <- 1..N ];

}  // Intergal_Array
```

Стоит заметить, что индекс массивов может иметь разные названия в этих уравнениях. Например где-то мы могли использовать букву `j` вместо буквы `i`. Также мы могли определить один из диапазонов как `(2-1)..(N-1+1)` и тому подобное.

Идея заключается в том, что записанная форма достаточно гибка. Уравнения не должны быть похожи друг на друга. Только размерность массивов должна соответствовать.

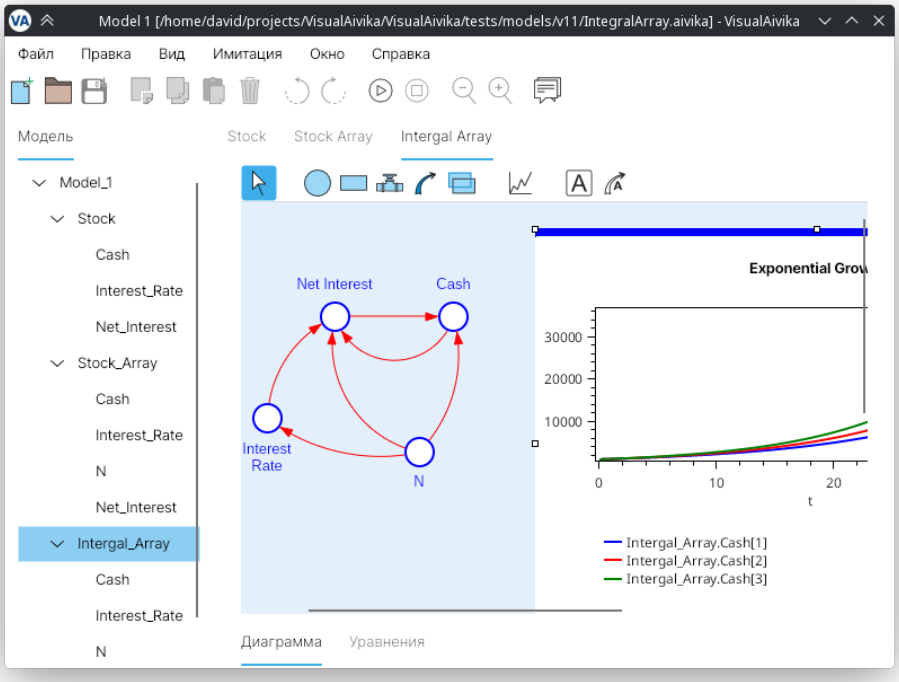


Рис. 7.2: Массив интегралов, представленный на диаграмме с помощью дополнительного элемента.

7.3 Массив накопителей интеграла

Массив накопителей интеграла — это уже совсем другая история. Синтаксис массива уже не настолько гибок. Индекс массива должен быть идентичным, а диапазоны массивов должны быть простыми.

Тот же массив интегралов показан на рисунке 7.3. Только сейчас интеграл определен с помощью элемента накопителя.

Эти уравнения похожи, но они используют особый искусственный синтаксис, который может напоминать об операциях с массивами в Matlab, Octave и NumPy. Так называемый дополнительный элемент пока не поддерживает этот синтаксис, но это может измениться в будущем.

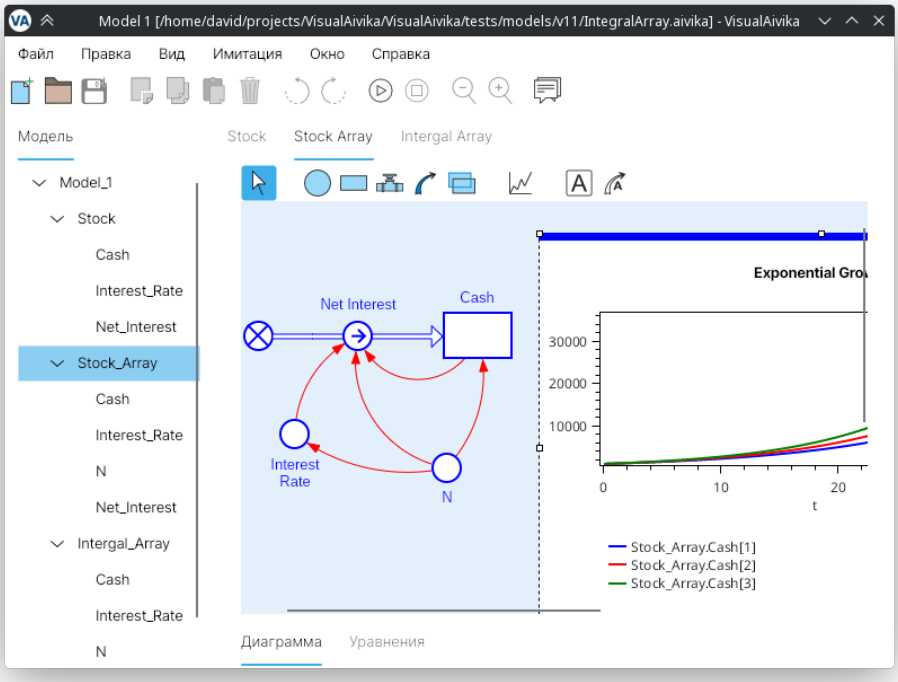


Рис. 7.3: Массив накопителей интеграла на диаграмме.

Соответствующие уравнения приведены ниже.

Пример

```
module Stock_Array {

  Cash = integ (Net_Interest, [ 1000 | i <- 1..N ]);
  Interest_Rate = [ 0.07 + i * 0.01 | i <- 1..N ];
  N = 3;
  Net_Interest = max (0,
    [ Cash[i] * Interest_Rate[i] | i <- 1..N ]);

}  // Stock_Array
```

Эти уравнения определяют ту же математику, но их форма совершенная иная. Обратите внимание на то, что индекс массива идентичен

во всех уравнениях. Также диапазон массива состоит только из целых чисел и переменных.

Что касается редактора уравнений, то соответствующие уравнения должны содержать массивы. Например, рисунок 7.4 показывает, как может быть определено начальное значение накопителя. Начальное значение — это массив, как и массивом является базовое значение для потока. В этом и заключается основная идея.

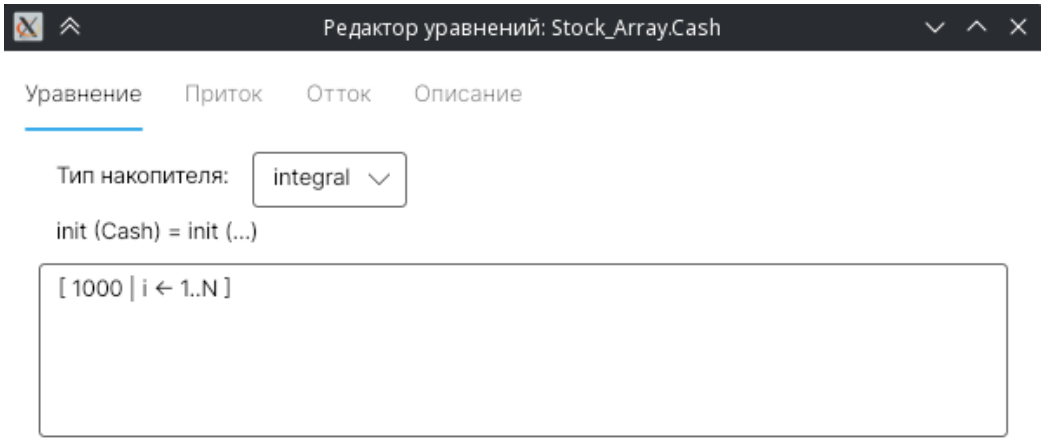


Рис. 7.4: Начальное значение массива накопителей интегралов.

7.4 Накопитель конвейера

Подобно интегралам конвейеры могут быть определены с помощью сооветствующей функции `conveyor` при использовании так называемого дополнительного элемента диаграммы. Однако существует более простой в использовании и интуитивный способ, когда конвейер и его потоки могут быть определены на диаграмме с помощью элементов накопителя и потока.

Следующий рисунок 7.5 показывает диаграмму для простой модели с одним конвейером, двумя притоками (входящими потоками) и тремя оттоками (исходящими потоками). На самом деле, оттоки специализированы, о чем мы вскоре поговорим.

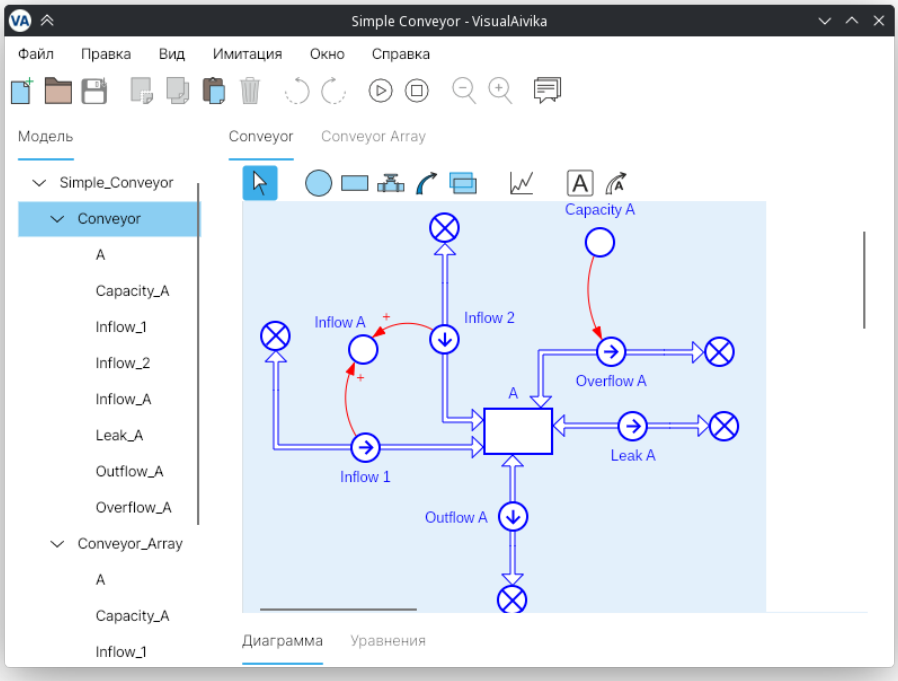


Рис. 7.5: Накопитель конвейера на диаграмме.

В настоящее время накопитель конвейера выглядит на диаграмме как накопитель интеграла. Хотя они действительно похожи, внешний вид может в будущем измениться. И интеграл, и конвейер — это аккумуляторы, где их потоки показывают, насколько быстро изменяется значение аккумулятора в течение модельного времени.

Прежде, чем перейти к более детальному рассмотрению элементов потока и накопителя, стоит показать соответствующие уравнения.

Пример

```
module Conveyor {

  conveyor A {

    inflow of Inflow_1, Inflow_2;
```

```
    outflow of Outflow_A;
    overflow of Overflow_A;
    leakflow of Leak_A;
    init = 0;
  }

  Capacity_A = 50;

  Inflow_1 =
    if (time >= 0-dt/2) and (time <= 2+dt/2) then 25 else 0;

  Inflow_2 =
    if (time >= 1-dt/2) and (time <= 3+dt/2) then 20 else 0;

  Inflow_A = Inflow_1 + Inflow_2;

  leakflow Leak_A from A {
    decay_rate = 0.05;
  }

  outflow Outflow_A from A {
    transit_time = 3;
  }

  overflow Overflow_A from A {
    capacity = Capacity_A;
  }

}  // Conveyor
```

Здесь уравнения для накопителя и потоков декларативны. Они говорят о том, что они определяют. Затем эти уравнения транслируются в соответствующие функции из раздела 5.23.

Накопитель конвейера определяет свое начальное значение и соответствующие потоки. Основной отток (outflow) определяет время прохождения, которое влияет на то, как долго входные данные содержатся во внутренней очереди конвейера. Конвейер может давать течь, и его ток утечки (leakflow) определяется соответствующим коэффициентом. Отток переполнения (overflow) возникает, когда конвейер имеет конечную емкость. Поэтому параметром оттока переполнения является значение емкости.

Например, рисунок 7.6 показывает, что мы можем определить емкость конвейера при редактировании уравнения для оттока перепол-

нения (overflow).

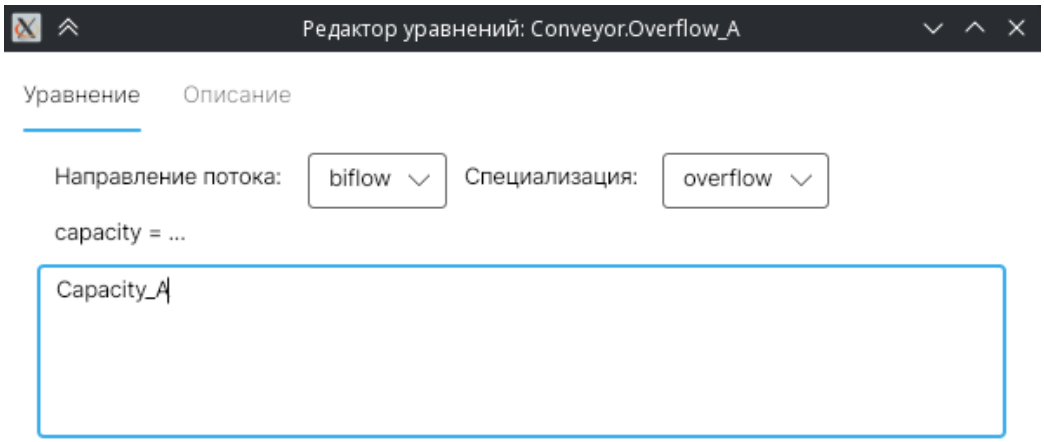


Рис. 7.6: Емкость накопителя конвейера.

Что касается самих уравнений, то мы создаем входящий поток, где переполнение возникает после значения модельного времени $1,25$. Также мы видим на рисунке 7.7, что основной отток имеет в точности ту задержку, что мы задали в уравнениях. Ток утечки сравнительно мал. Часть входных данных теряется при переполнении, из-за чего форма основного оттока отличается от формы входных данных.

Важно заметить, что форма значений конвейера и его потоков на графике довольно стабильна относительно небольших изменений параметра шага времени интегрирования `dt`. Это связано с тем, что потоки являются значениями интенсивности подобно производным для интеграла.

7.5 Массив накопителей конвейера

Как и прежде, мы могли бы определить массивы конвейеров с помощью синтаксиса массивов и встроенной функции `conveyor`. Однако здесь мы расширим предыдущую модель так, что элемент накопителя будет указывать уже на массив конвейеров. Диаграмма остается в точности такой же, какой она была показана на рисунке 7.5. Мы ме-

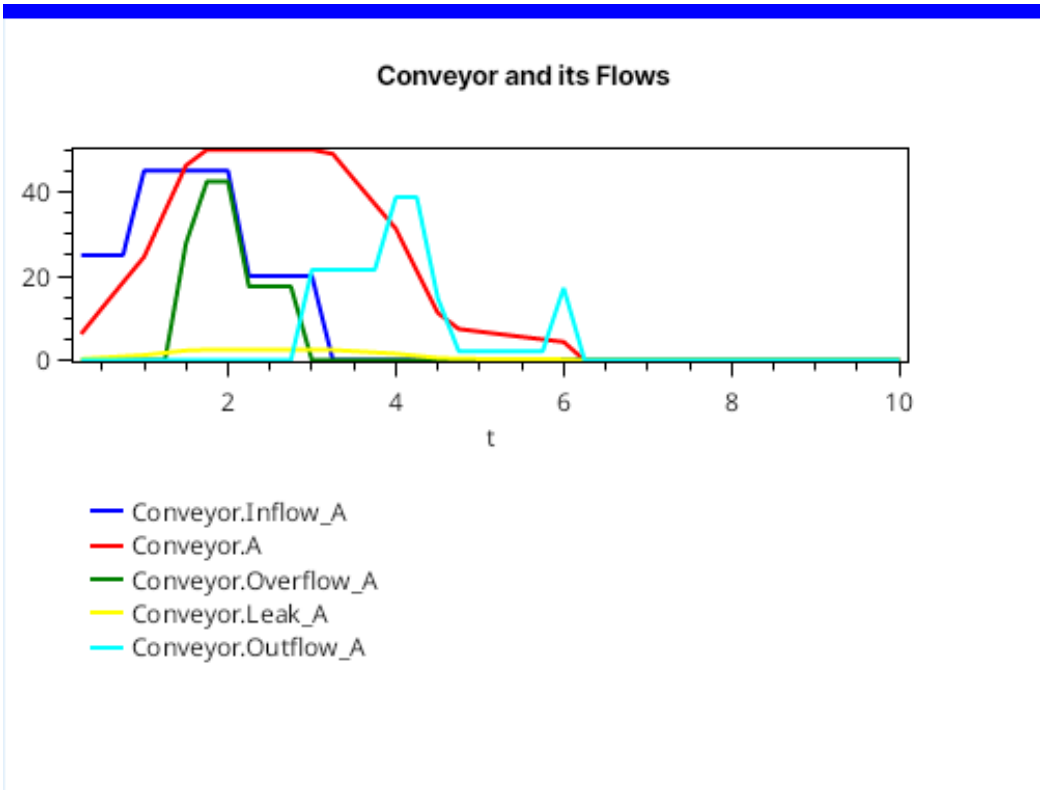


Рис. 7.7: Значения конвейера и его потоков.

няем только уравнения. Вместо скалярных значений мы определяем массивы в редакторе уравнений для соответствующих параметров.

Пример

```
module Conveyor_Array {

  conveyor A {

    inflow of Inflow_1, Inflow_2;
    outflow of Outflow_A;
    overflow of Overflow_A;
    leakflow of Leak_A;
    init = [ 0 | i <- 1..3 ];
  }

  Capacity_A = [ 50 + 3*(i - 1) | i <- 1..3 ];
```

```
  Inflow_1 =
    [ if (time >= 0-dt/2) and (time <= 2+dt/2)
        then 25 else 0 | i <- 1..3 ];

  Inflow_2 =
    [ if (time >= 1-dt/2) and (time <= 3+dt/2)
        then 20 else 0 | i <- 1..3 ];

  Inflow_A = [ Inflow_1[i] + Inflow_2[i] | i <- 1..3 ];

  leakflow Leak_A from A {
    decay_rate = [ 0.05 | i <- 1..3 ];
  }

  outflow Outflow_A from A {
    transit_time = [ 3 | i <- 1..3 ];
  }

  overflow Overflow_A from A {
    capacity = [ Capacity_A[i] | i <- 1..3 ];
  }

}  // Conveyor_Array
```

Здесь мы изменили только значения емкостей. Как результат, изменились и значения конвейеров, соответственно, как показано на рисунке 7.8.

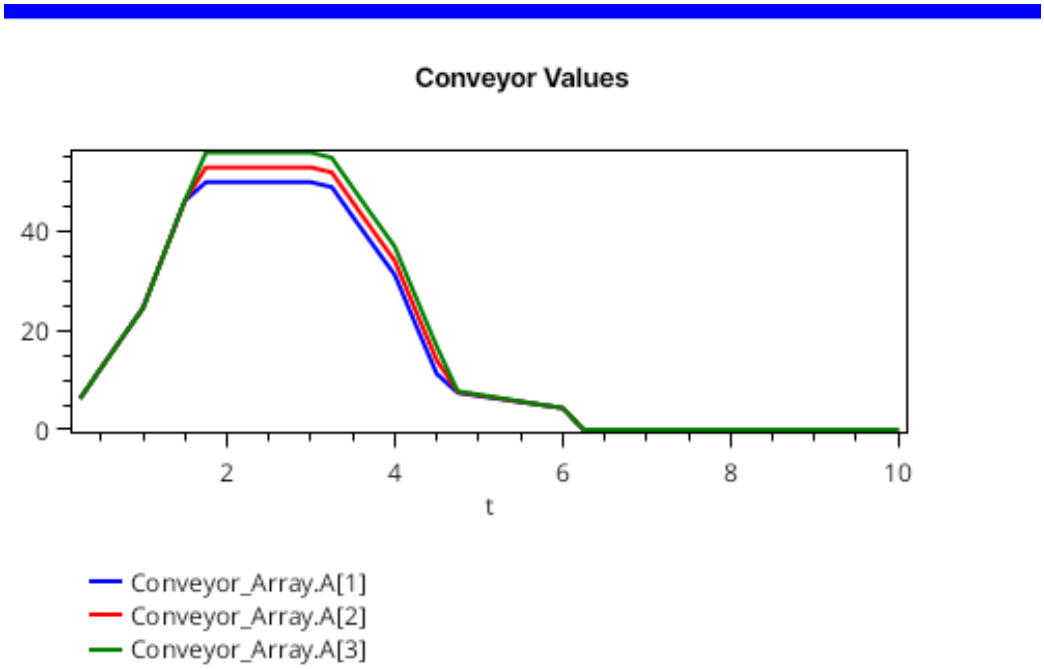


Рис. 7.8: Значения массива конвейеров.

Глава 8

Частичное моделирование

Частичное моделирование полезно, когда мы собираемся создать сложную имитационную модель из маленьких частей и шаг за шагом. Мы можем определить только маленькую часть целой модели, где только эта часть будет использована для имитации, и только она будет проверяться на корректность.

Для определения новой под-модели мы должны задать ее и затем выбрать из главного меню, как показано на рисунке 8.1.

Здесь пункт меню *Вся модель* говорит о том, что вся модель рассматривается для имитации. Это режим по умолчанию. Однако если мы выберем под-модель, то тогда только та часть будет рассматриваться.

Соответствующие уравнения окрасятся в светло-серый, если они не используются. Тем не менее, если на уравнение все же ссылаются, но оно само не включено в заданную под-модель, то тогда только начальное значение соответствующей переменной используется. Цвет такого уравнения будет просто серым.

Это работает благодаря тому, что большую часть модели можно отсечь, если использовать только начальные значения некоторых переменных. Более того, некоторые переменные могут быть даже не определены.

Что касается самого редактора под-моделей, то он показан на рисунке 8.2.

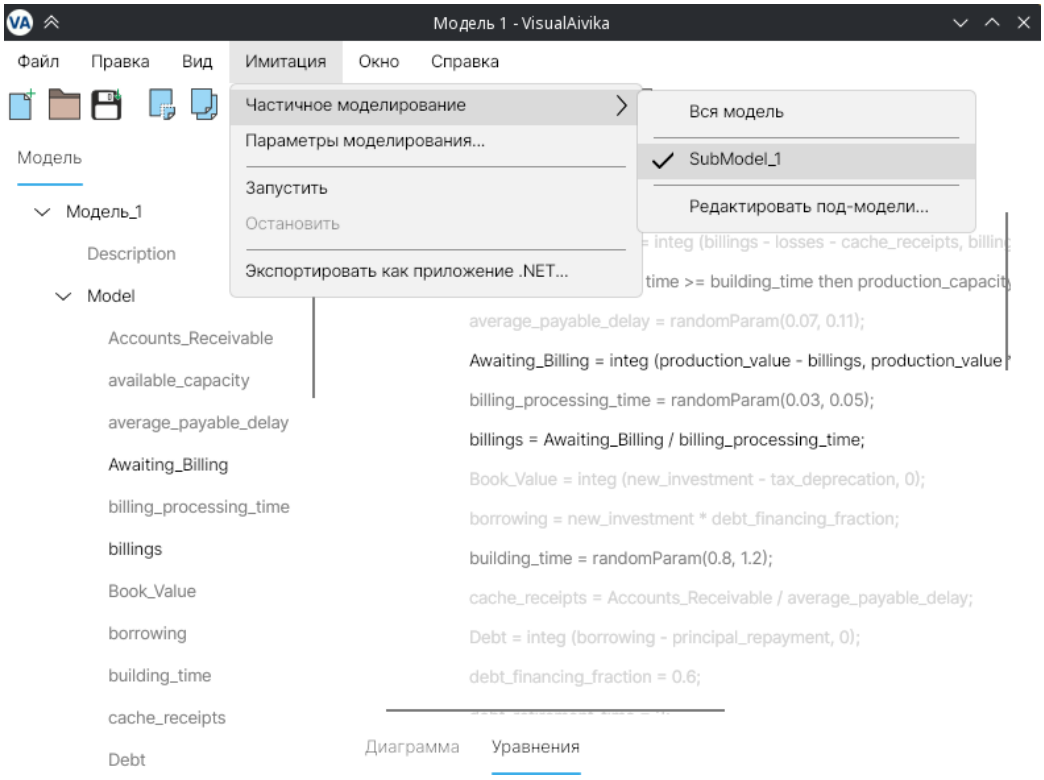


Рис. 8.1: Меню для частичного моделирования.

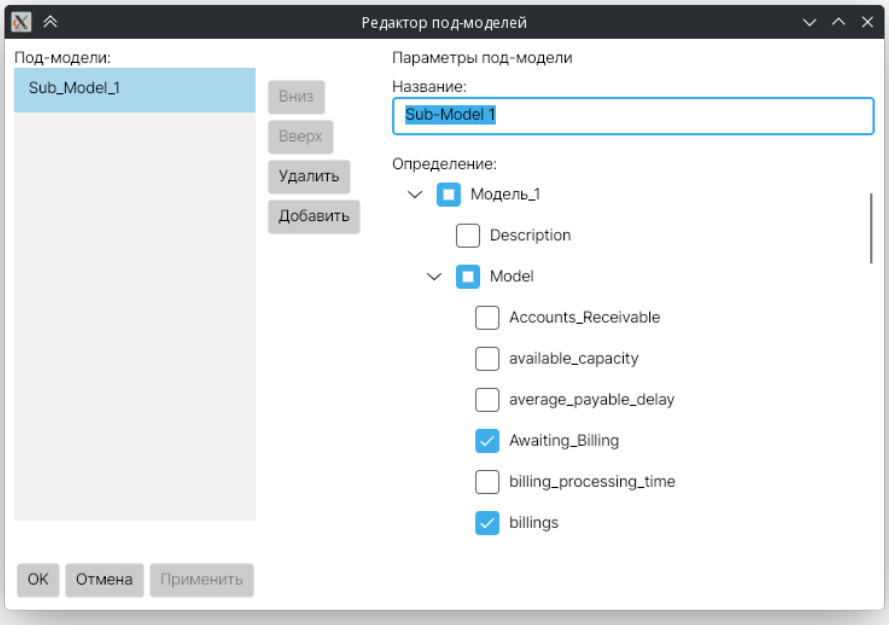


Рис. 8.2: Редактор под-моделей для частичного моделирования.

Глава 9

Гибридное моделирование

Для демонстрации возможности гибридного моделирования мы рассмотрим следующую задачу. Мы берем некоторую входную интенсивность, которая может зависеть от дифференциальных уравнений, например. Затем мы создаем соответствующий пуассоновский процесс по заданной интенсивности. Это будет соответствовать некоторому дискретному сигналу, по которому мы можем инициировать обработку транзактов. Для краткости, обработка транзактов будет простейшей — мы просто будем подсчитывать транзакты для оценки их интенсивности. Эта оценка интенсивности будет слишком дискретной. Мы возьмем экспоненциальное сглаживание, а затем сравним полученную на выходе интенсивность с соответствующей исходной интенсивностью.

Наша модель будет иметь следующую диаграмму, изображенную на рисунке 9.1.

Соответствующие уравнения приведены ниже.

Пример

```
dt = 0.25;

Input_Rate_A = 1 + 0.5 * sin(time);

Signal_A = Signal.exponential(1 / Input_Rate_A);
Counter_A = TransactCounter.create();
Block_A = Block.count(Counter_A) >>> Block.terminate;

Runner = do! Block.runBySignal(Signal_A, Block_A);
```

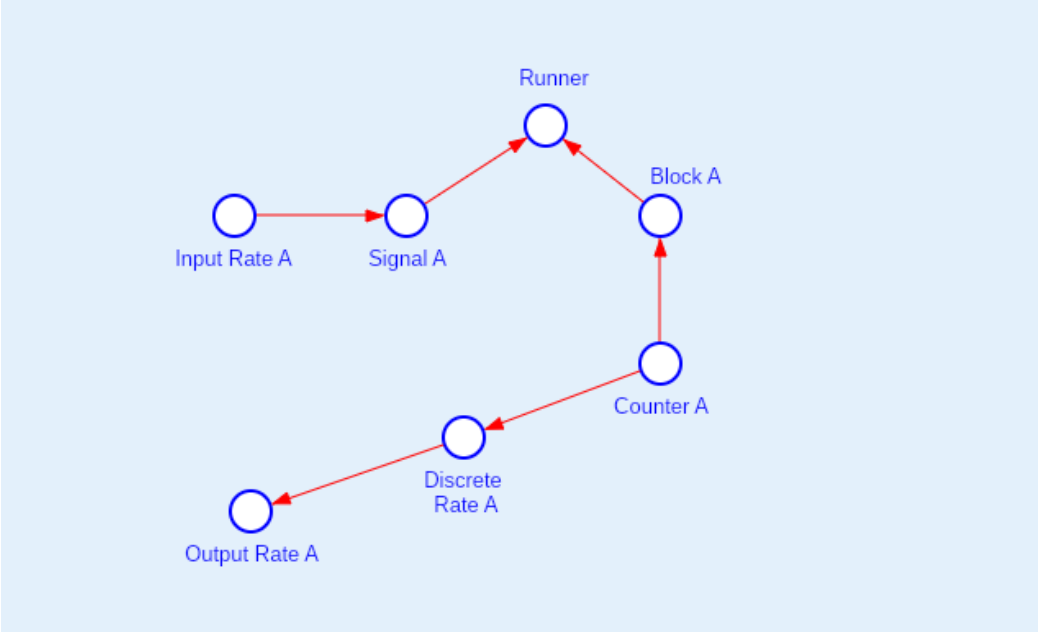


Рис. 9.1: Диаграмма гибридной модели.

```
Discrete_Rate_A = TransactCounter.rate(Counter_A);
Output_Rate_A = smooth(Discrete_Rate_A, 0.2, 1);
```

Переменная `Input_Rate_A` задает входную интенсивность. Результат называется `Output_Rate_A`. Они должны соответствовать друг другу.

Если мы сделаем одиночный имитационный запуск, то мы можем увидеть нечто непонятное, похожее на рисунок 9.2, где мы видим некоторую реализацию случайного процесса. Его вид меняется от запуска к запуску.

Однако, чтобы увидеть истинную закономерность случайного процесса, мы должны сделать множество запусков, как показано на рисунке 9.3, где мы видим, что тренд выходной интенсивности имеет небольшую задержку, но он все же соответствует входной интенсивности.

Это все остается рабочим, если мы будем использовать дискретную входную интенсивность.

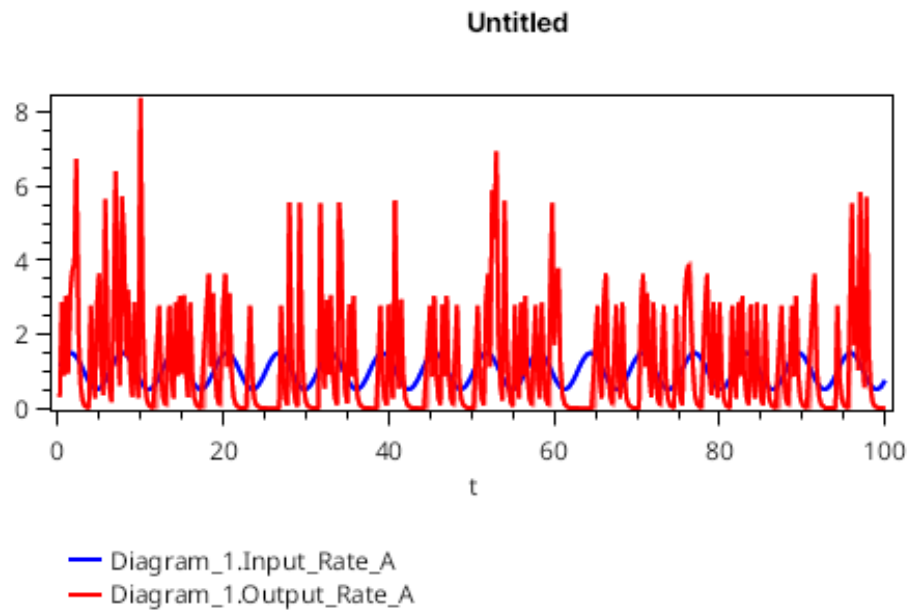


Рис. 9.2: Сравнение исходной и выходной интенсивностей для одиночного запуска.

Пример

```
Input_Rate_A =
  if time < 5 + dt/2
    then 1
    else (if time < 15 + dt/2 then 5 else 1);
```

Итоговая выходная интенсивность будет все же иметь задержку, как показано на рисунке 9.4.

В то же самое время истинная дискретная выходная интенсивность без экспоненциального сглаживания будет ближе к исходной входной интенсивности, но сглаживающая функция все же сделала бы оценку интенсивности более подходящей для использования ее в дифференциальных уравнениях. Оба подхода имеют свои достоинства и недостатки. Второй график показан на рисунке 9.5.

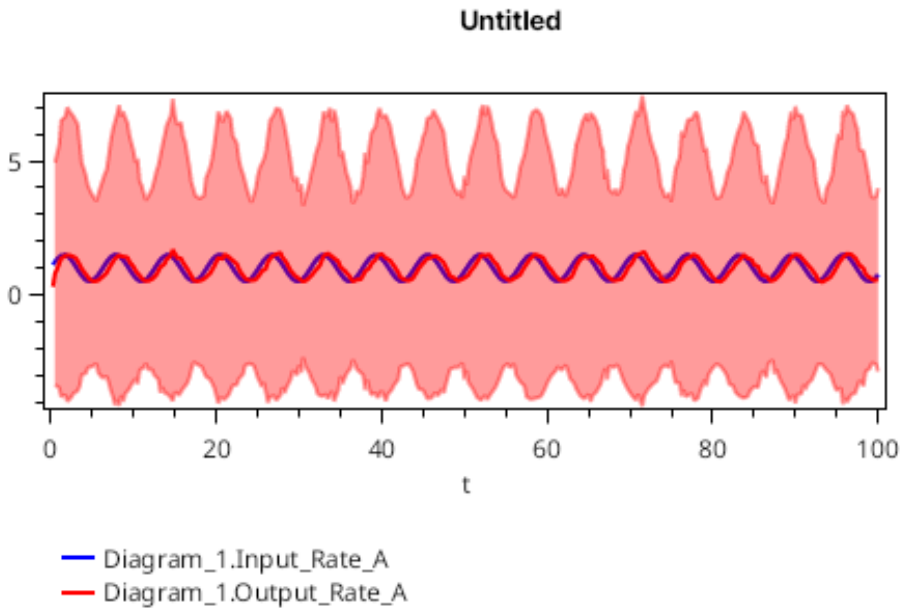


Рис. 9.3: Сравнение исходной и выходной интенсивностей для 1000 запусков.

Более того, если мы уменьшим временной интервал `dt`, то тогда все оценки станут еще ближе к исходной входной интенсивности.

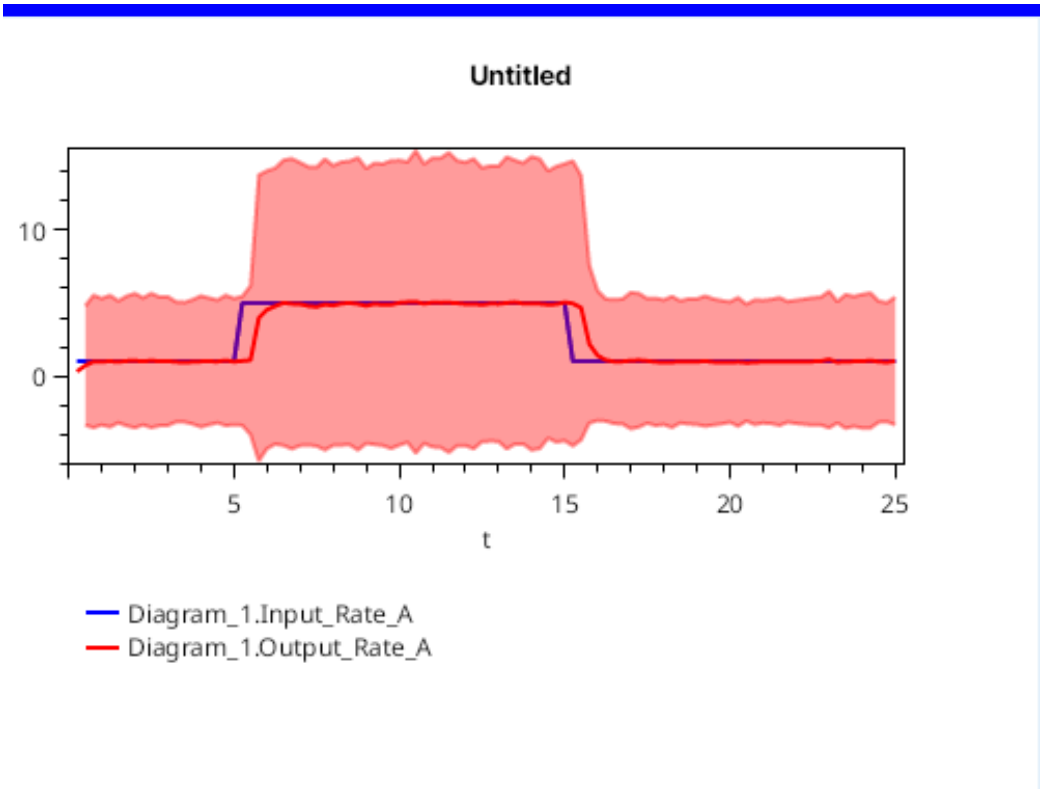


Рис. 9.4: Сравнение исходной дискретной и выходной интенсивностей.

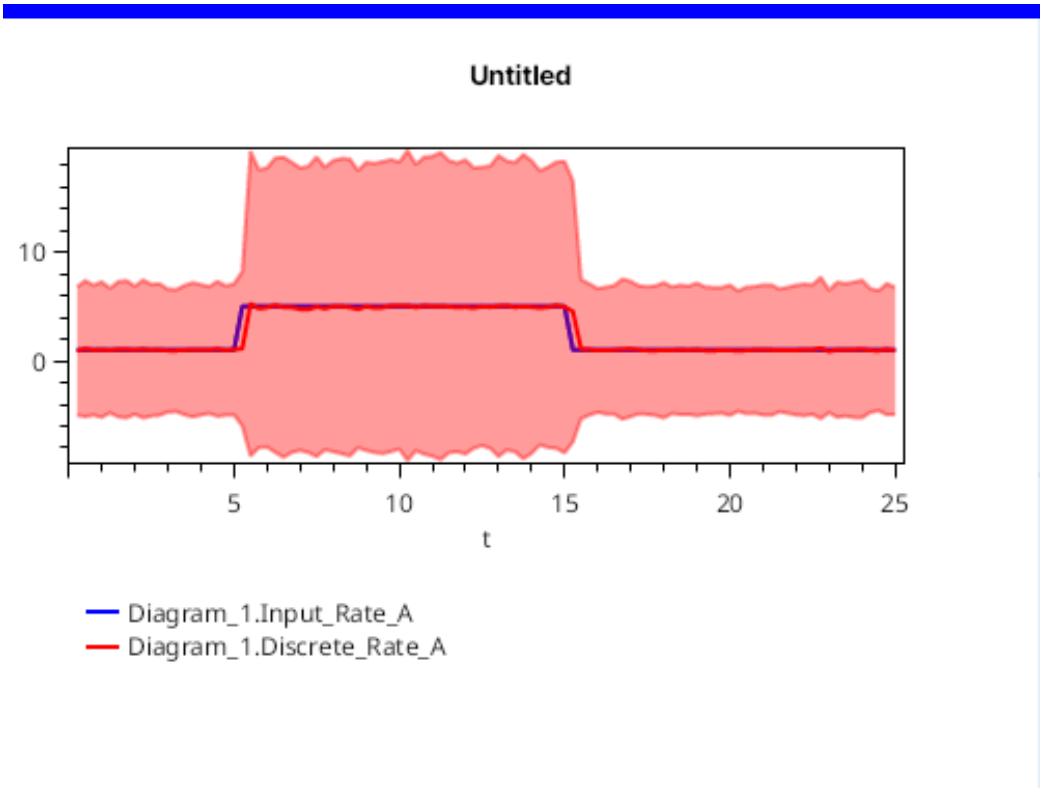


Рис. 9.5: Сравнение исходной дискретной и выходной интенсивностей без экспоненциального сглаживания.

Глава 10

Экспорт приложений .NET

Версия программы «ВизуальнаяАйвика» с компилятором уравнений позволяет вам экспортировать ваши имитационные модели как приложения .NET. Такие экспортированные приложения будут зависеть от кода программных библиотек, которые доступны в исходном виде под соответствующими лицензиями. Это означает, что экспортированные приложения могут быть собраны и запущены без самой программы «ВизуальнаяАйвика». Программа «ВизуальнаяАйвика» может быть использована только для моделирования и экспорта.

Эта возможность особенно полезна, если вы собираетесь расширить вашу имитационную модель с помощью произвольного кода .NET, который может быть включен в имитацию. Пожалуйста, обратитесь к главе 11 для дополнительной информации.

Чтобы экспортировать имитационную модель как приложение .NET, пожалуйста, выберите меню *Имитация / Экспортировать как приложение .NET*. Затем выберите каталог, куда «ВизуальнаяАйвика» запишет приложение. Перед этим «ВизуальнаяАйвика» может попросить вас сохранить модель в случае необходимости.

Итоговое приложение .NET будет экспортировано в простейшей форме, где оно отображает результаты в конечной точке времени. Однако сам код достаточно общий. Он может быть адаптирован для разных целей. Пожалуйста, для дополнительной информации прочитайте руководство *IronAivika: Simulation Library for .NET*.

Чтобы запустить тестовое приложение, вам следует переключиться в соответствующий каталог, куда приложение было экспортировано.

Затем вы можете ввести в консоли[1]:

```
$ dotnet build
$ dotnet run
```

Это работает, если вы увидете что-то похожее на

```
----------

// time
t = 10

Diagram_1.A = 220,19641241130046
```

Здесь показано конечное значение для некоторой переменной `Diagram_1.A`. Вы можете увидеть другой вывод для вашей собственной имитационной модели.

Это пластилин, из которого вы можете вылепить любую фигуру.

[1] В системе Windows значок приглашения будет другим.

Глава 11

Внешние модули

«ВизуальнаяАйвика» позволяет вам расширять ваши имитационные модели с помощью вызовов во время имитации произвольного программного кода на .NET. Это особенно полезно, если вы собираетесь экспортировать ваши имитационные модели как приложения .NET, что было описано в главе 10.

Для написания и использования внешних модулей, пожалуйста, ознакомьтесь с руководством *VisualAivika Manual: Creating External Modules*.

Пример

```
F =
  [<assembly("lib/DelayBlockFunction.dll")>]
  [<class(VisualAivika.Demo.DelayBlockFunction)>]
  extern fun (A: Block): Block;

Y1 = F(Block.advance(10) >>> Block.terminate);
```

Здесь тип класса `DelayBlockFunction` может быть определен на языках F# или C#. Тип класса должен быть расположен в сборке `DelayBlockFunction.dll`.

Внешние модули могут определять функции и сами модули. Последние могут иметь множество входных и выходных параметров. Представленная в примере функция — это лишь частный случай модуля.

Приложение А

Детали реализации прибора

Эти разделы приведены в качестве справочного материала для более глубокого понимания того, как моделируется модель.

А.1 Вытеснение прибора

Ниже описана логика, лежащая в основе вычисления `Block.preempt`.

Опциональный параметр `priorityMode` указывает, используется ли режим приоритета или прерывания (по умолчанию). Опциональный параметр `transfer` может определять цепочку блоков, в которую передается транзакт в случае вытеснения. Параметр `removalMode` указывает, используется ли режим удаления (по умолчанию не используется).

Концептуально, вычисление `Block.preempt` пытается имитировать поведение блока `PREEMPT` из языка моделирования GPSS, хотя реализация в программе «ВизуальнаяАйвика» основана на совершенно других принципах, которые уходят корнями в функциональное программирование.

У прибора может быть только один владелец, т.е. транзакт, которому принадлежит прибор, или у прибора вообще нет владельца. Кроме того, у прибора есть три очереди: *Цепочка задержек*, *Цепочка прерываний* и *Цепочка ожидания*. Цепочка задержек и цепочка ожидания аналогичны FIFO, в то время как цепочка прерываний близка к LIFO, но эти очереди используют приоритеты. В очередях FIFO хранятся соответствующие вычисления транзактов, а также сами транзакты.

Алгоритм, применяемый в вычислении `Block.preempt`, заключается в следующем.

1. Если у прибора нет владельца, то текущий транзакт становится владельцем, но как владелец, который помечен как *не прерывающий*.

2. В противном случае, если параметр `priorityMode` задан как `false` (по умолчанию), а текущий владелец является *прерывающим*, то текущий транзакт помечается как прерывающий, и после этого вычисление транзакта добавляется в цепочку ожидания прибора с приоритетом транзакта.

3. В противном случае, если параметр `priorityMode` задан как `true` и приоритет транзакта ниже (меньше), чем приоритет владельца, текущий транзакт помечается как прерывающий, а его вычисление добавляется в цепочку задержек прибора с приоритетом транзакта.

4. В противном случае, если параметр `removalMode` равен `false` (по умолчанию), то текущий транзакт заменяет владельца. Текущий транзакт помечается как прерывающий. Предыдущий владелец вытесняется и добавляется в цепочку прерываний со своим приоритетом.

   В следующий раз, когда предыдущий владелец вернется из цепочки прерываний, и если параметр `transfer` не указан (по умолчанию), то этот транзакт снова захватит прибор и продолжит свое выполнение из блока, где он был вытеснен.

   В противном случае, если параметр `transfer` указывает на цепочку блоков, возвращенный владелец будет переведен в соответствующую цепочку блоков, где указанный атрибут транзакта будет содержать интервал времени, оставшийся при удержании транзакта, начиная со времени, в которое прежний владелец был вытеснен.

5. Если другие случаи не выполняются, и, следовательно, параметр `removalMode` явно указан как `true`, то текущий транзакт становится новым владельцем и помечается как прерывающий. Предыдущий владелец вытесняется и переводится в цепочку блоков,

возвращаемую указанным параметром `transfer`, который должен определять вычисление цепочки блоков. Время, оставшееся для удержания прежнего владельца, передается через соответствующий атрибут транзакта. Теперь параметр `transfer` является обязательным. Цепочка прерываний не используется. Предыдущий владелец удаляется в момент текущего модельного времени.

A.2 Захват прибора

Применяется следующий алгоритм для вычисления `Block.seize`.

1. Если у объекта есть уже другой владелец, то вычисление транзакта блокируется до тех пор, пока этот другой владелец не освободит или не вернет прибор. В таком случае текущий транзакт помечается как не прерывающий, и он вместе с вычислением добавляется в цепочку задержек прибора с приоритетом транзакта.

2. В противном случае, если у прибора нет владельца, текущий транзакт становится владельцем прибора. Этот владелец помечается как не прерывающий, и транзакт продолжает свое выполнение.

A.3 Освобождение прибора

Вычисление `Block.release` должно соответствовать вычислению `Block.seize`. В остальном, вычисление освобождения аналогично процедуре возврата прибора, описанной далее.

A.4 Возврат прибора

Вычисление `Block.return` должно соответствовать `Block.preempt`. В остальном, вычисления по освобождению и возврату прибора имеют схожее поведение.

Владельцем прибора должен быть текущий транзакт. Алгоритм выбора другого владельца следующий.

1. Если цепочка ожидания прибора не пуста, и вычисление транзакта-кандидата из очереди не отменено, то соответствующий транзакт удаляется из очереди и становится владельцем. Цепочка ожидания близка к FIFO, но использует приоритеты.

   Если вычисление транзакта-кандидата было отменено, то транзакт также удаляется из очереди, и алгоритм повторяется с самого начала.

2. В противном случае, если цепочка прерываний прибора не пуста, и вычисление транзакта-кандидата из очереди не отменено, то соответствующий транзакт удаляется из очереди и выбирается в качестве нового владельца.

   Теперь, если транзакт был прерван вместе с указанием параметра `transfer`, то этот параметр используется для переноса вычисления транзакта в соответствующую цепочку блоков, возвращаемую параметром `transfer`. Предыдущее вычисление отменяется, и вместо него создается новое, которое уже запускается с указанной цепочкой блоков. Очередь цепочки прерываний близка к LIFO, но использует приоритеты.

   Если параметр `transfer` не был указан, то транзакт продолжает свое выполнение с того вычисления, при котором транзакт был вытеснен.

   Если вычисление транзакта-кандидата было отменено, то транзакт также удаляется из очереди, и алгоритм повторяется с самого начала.

3. В противном случае, если цепочка задержек прибора непустая, и вычисление транзакта-кандидата из очереди не отменено, соответствующий транзакт удаляется из очереди и становится владельцем. Цепочка задержек близка к FIFO, но использует приоритеты.

   Если вычисление транзакта-кандидата было отменено, то транзакт также удаляется из очереди, и алгоритм повторяется с самого начала.

4. Если все остальные случаи отпадают, то это значит, что все три очереди пусты и, следовательно, у прибора нет владельца.

Литература

[1] Berkeley Madonna. http://www.berkeleymadonna.com, 2024. Accessed: 20-July-2024.

[2] A.A.B. Pritsker and J.J. O'Reilly. *Simulation with Visual SLAM and AweSim*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 1999.

[3] Thomas Schriber. *Simulation using GPSS*. Wiley, 1974.

[4] AnyLogic Software. http://www.anylogic.com, 2014. Accessed: 11-July-2026.

[5] Vensim Software. http://vensim.com, 2024. Accessed: 20-July-2024.