Руководство по VisualAivika

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

22 апреля 2025 г.

Оглавление

Глава 1

Введение

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 демонстирует 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, VisualAivika 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, VisualAivika поддерживает массивы и индексы. Рисунок 1.5 показывает график, который соответствует некоторому мас-

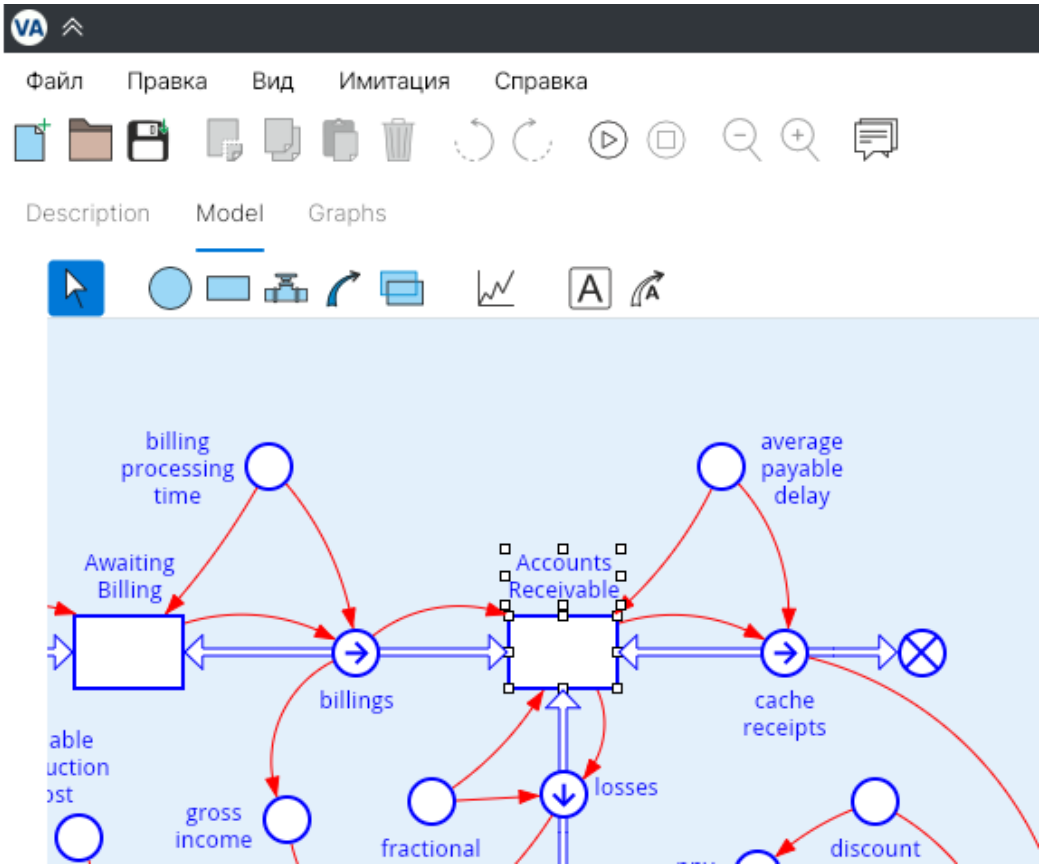

Рис. 1.1: Редактирование потоковой диаграммы SFM.

сиву.

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

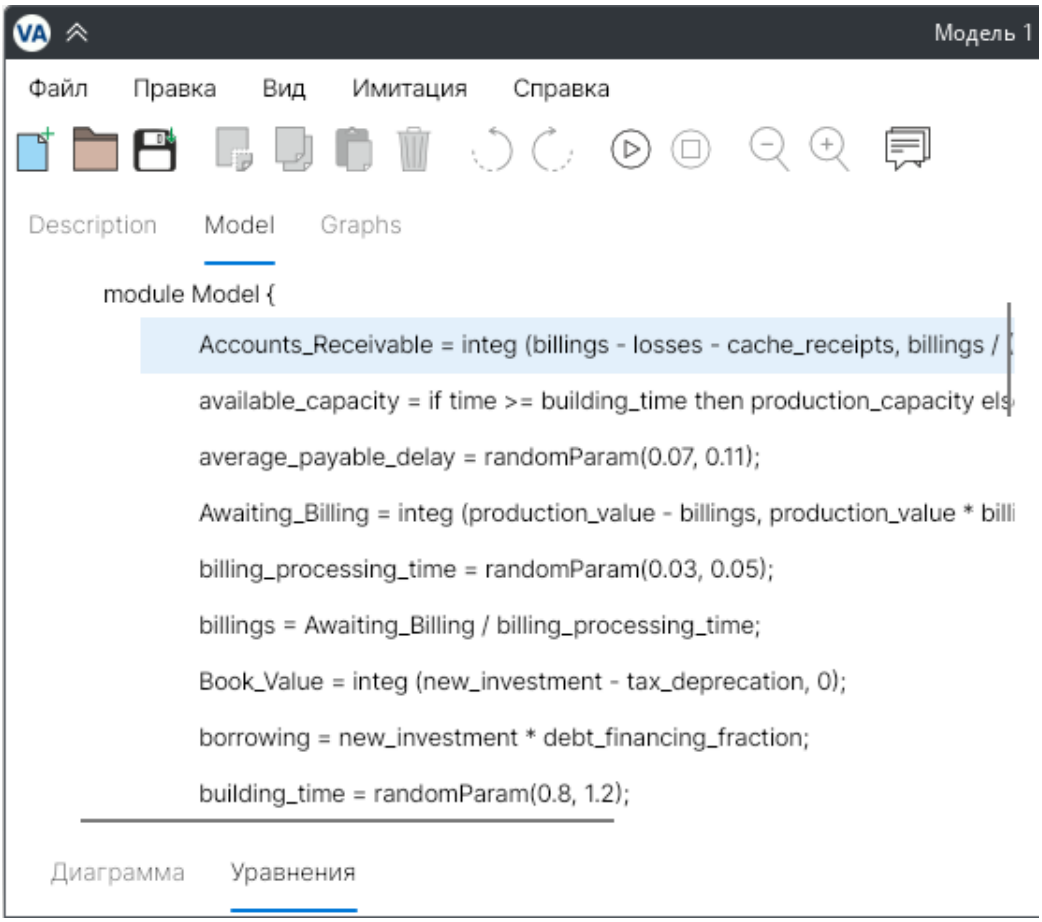

Рис. 1.2: Панель уравнений.

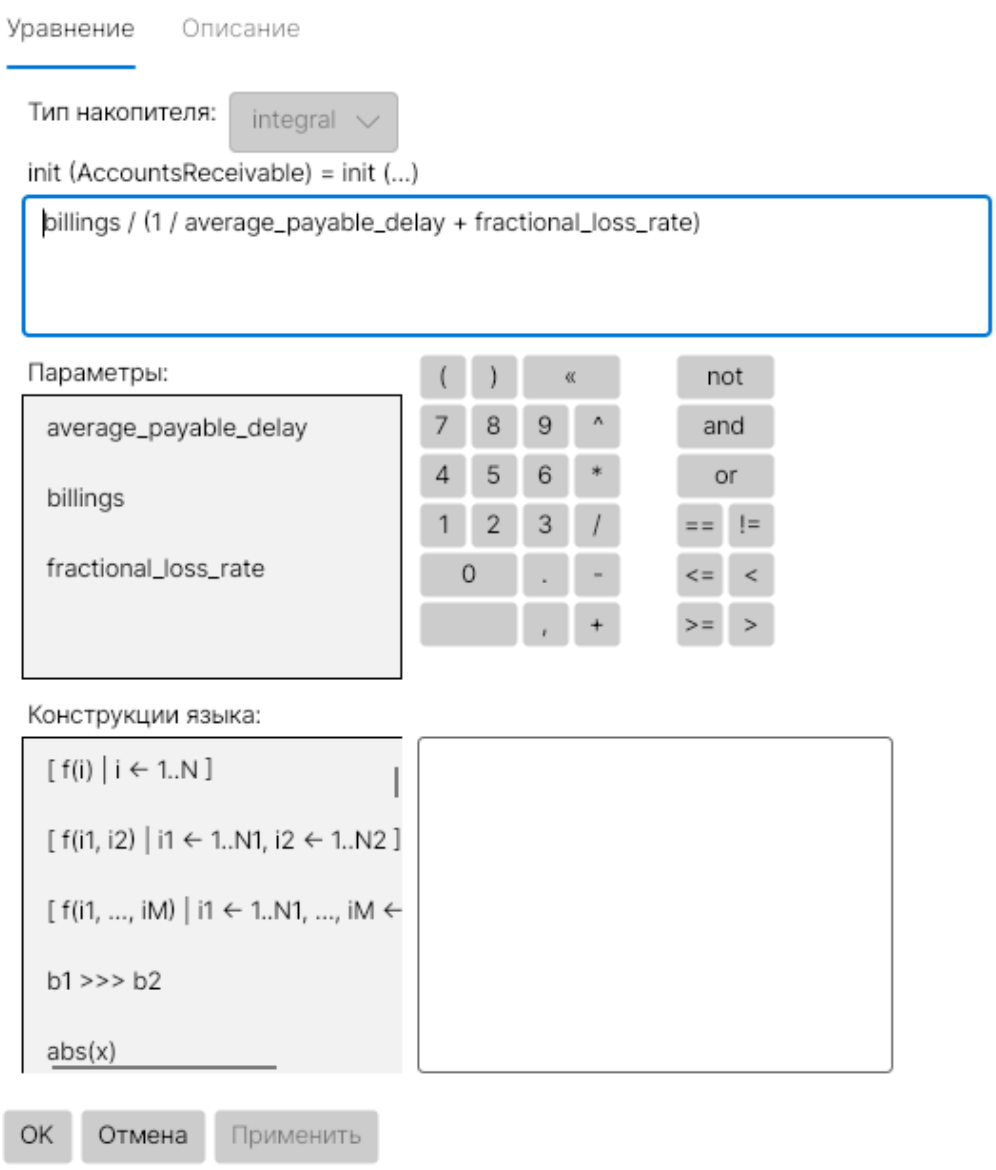

Рис. 1.3: Панель редактора уравнений.

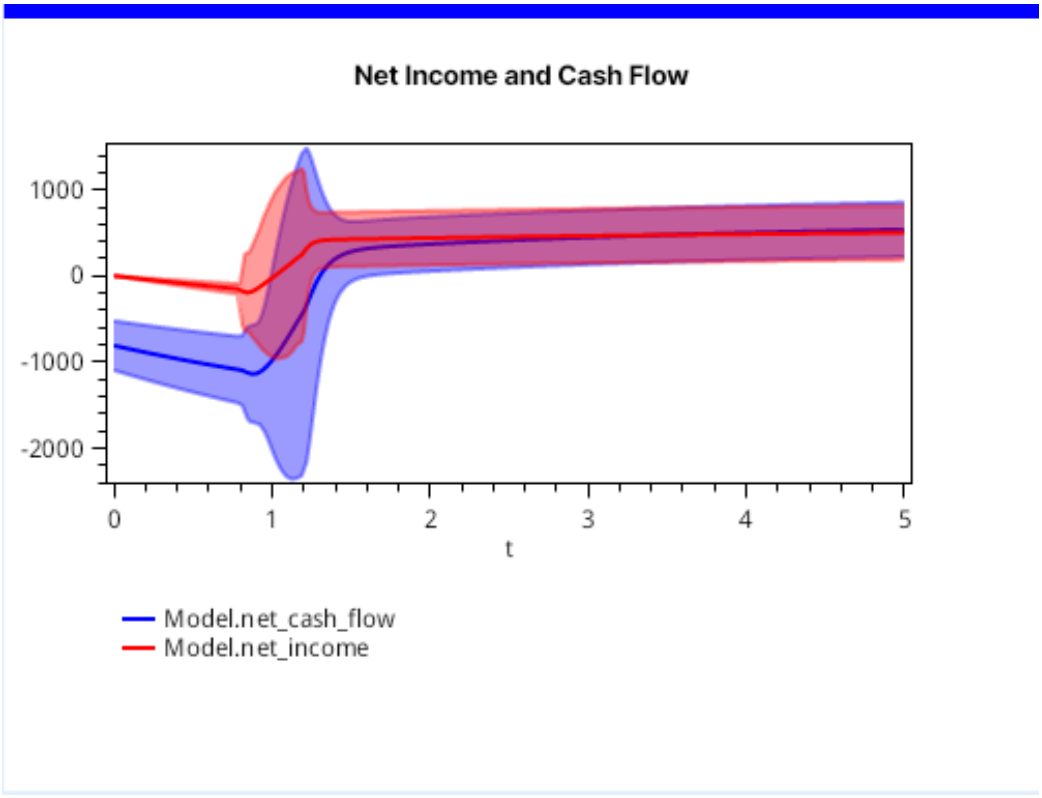

Рис. 1.4: График трендов с доверительными интервалами для анализа чувствительности.

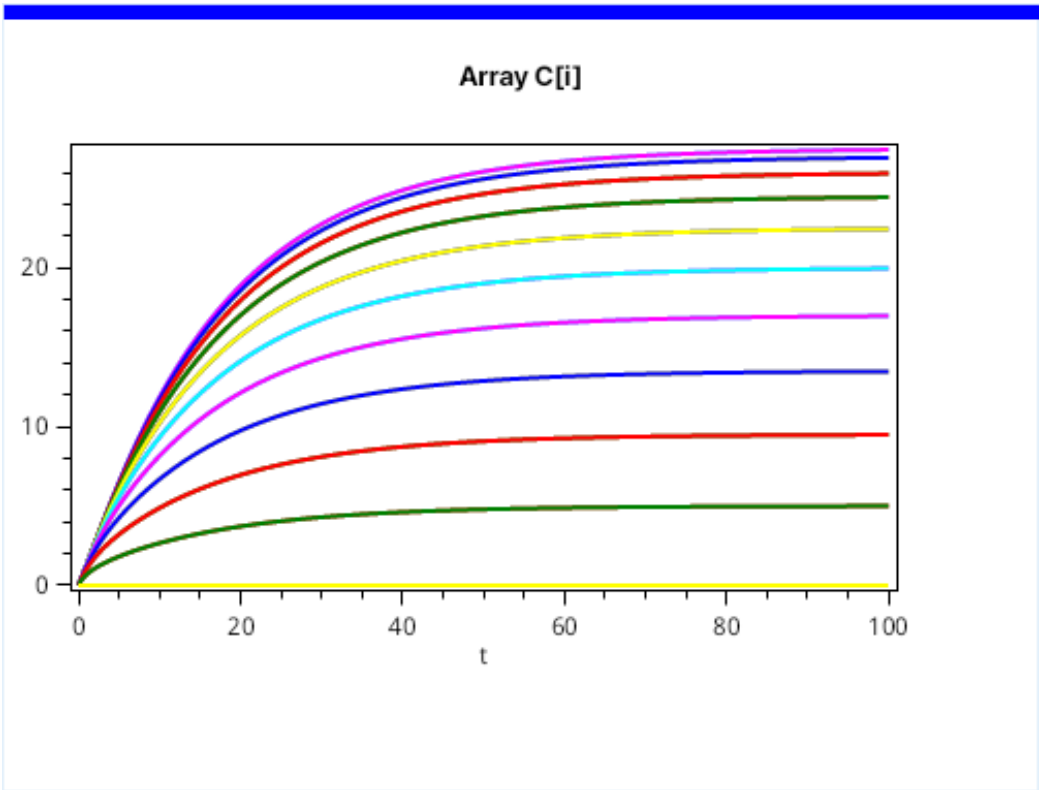

Рис. 1.5: Отображение массивов на графике.

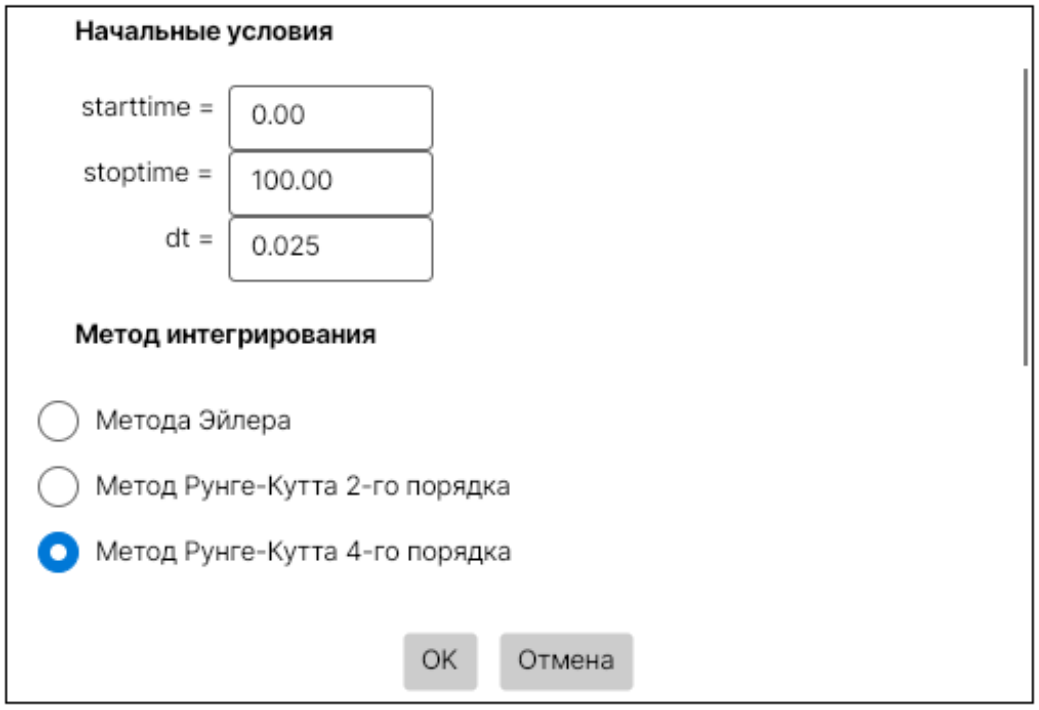

Рис. 1.6: Определение параметров моделирования.

Глава 2

Как создать простую модель

Например, мы можем воспроизвести следующие уравнения из модели ”5-Minute Tutorial” для программного инструмента 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.

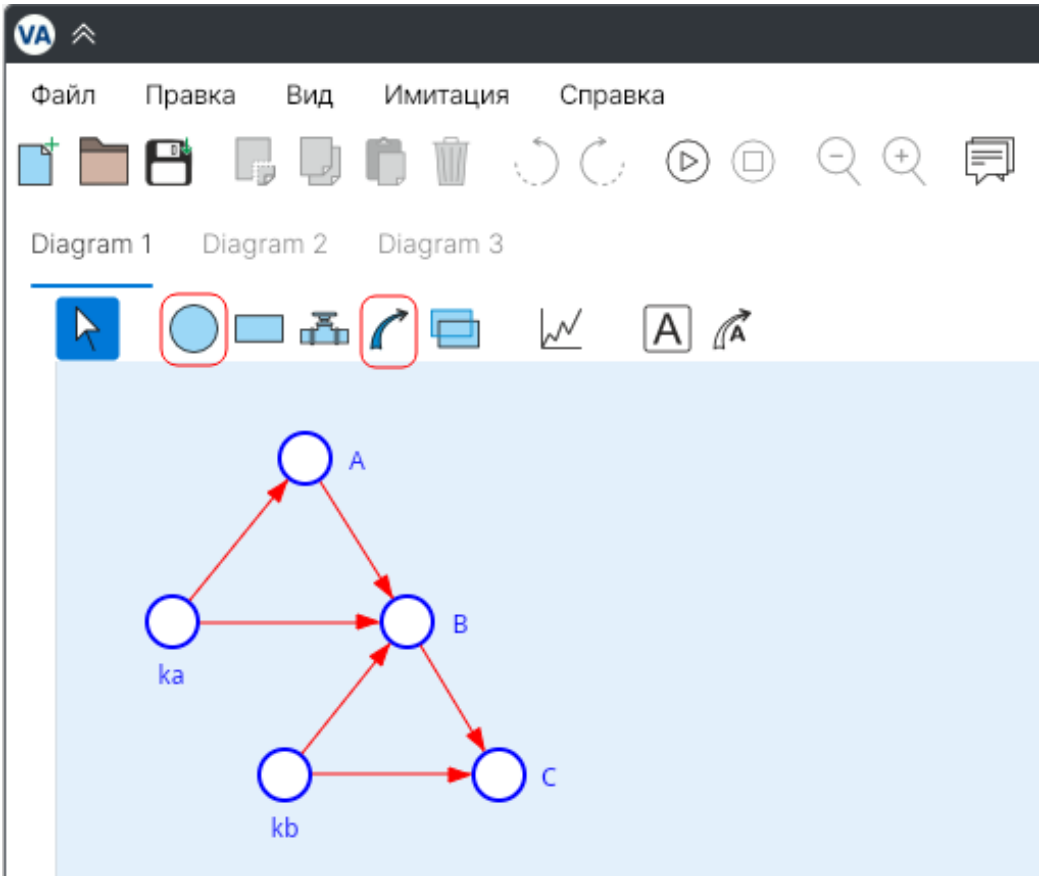

Рис. 2.1: Диаграмма состоит из будущих интегралов.

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 инстру-

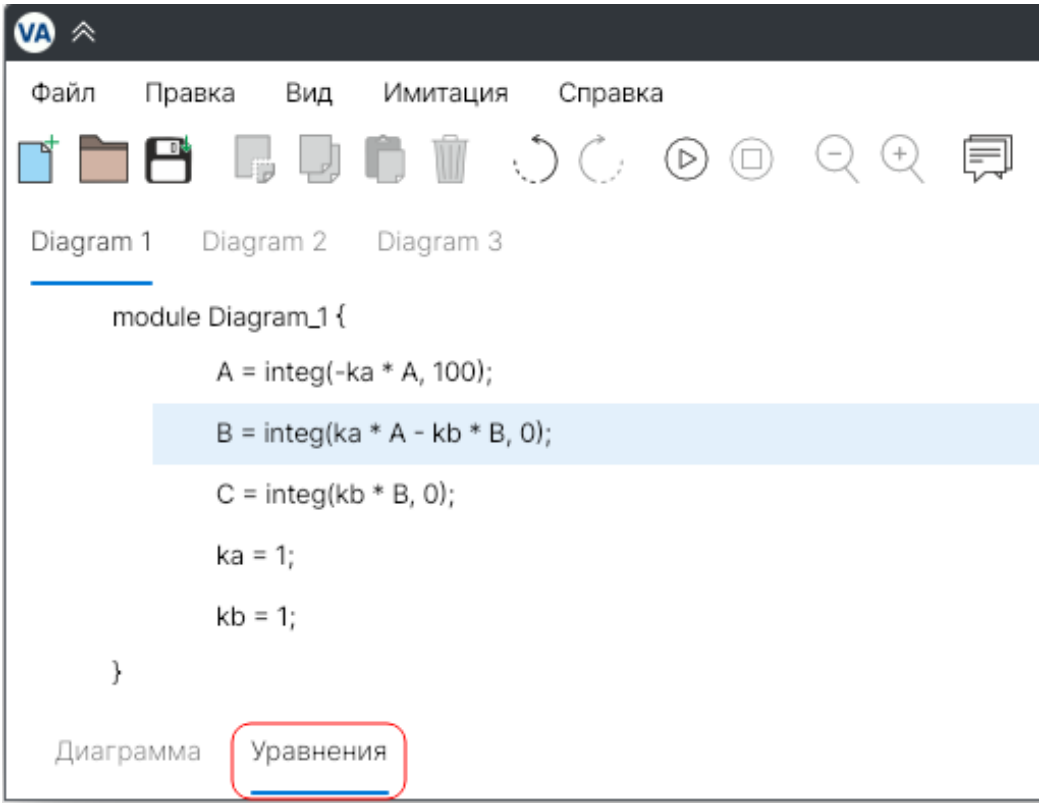

Рис. 2.2: Уравнения, состоящие из интегралов.

ментов 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).

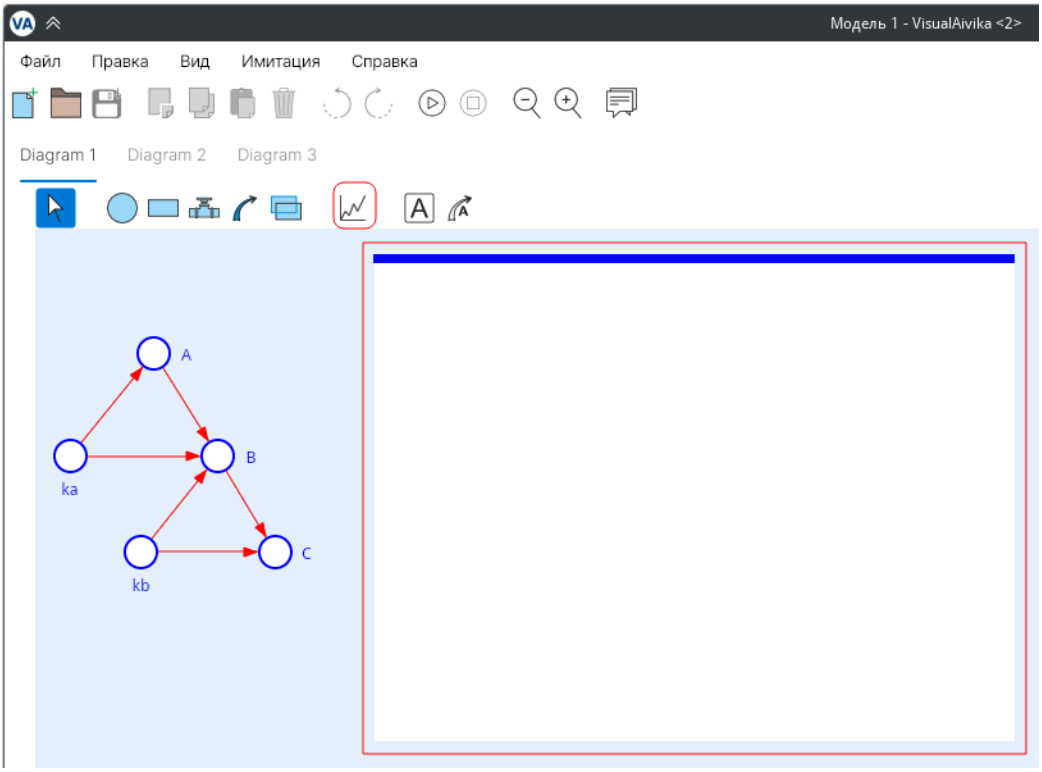

Рис. 2.3: После того, как график добавлен на диаграмму.

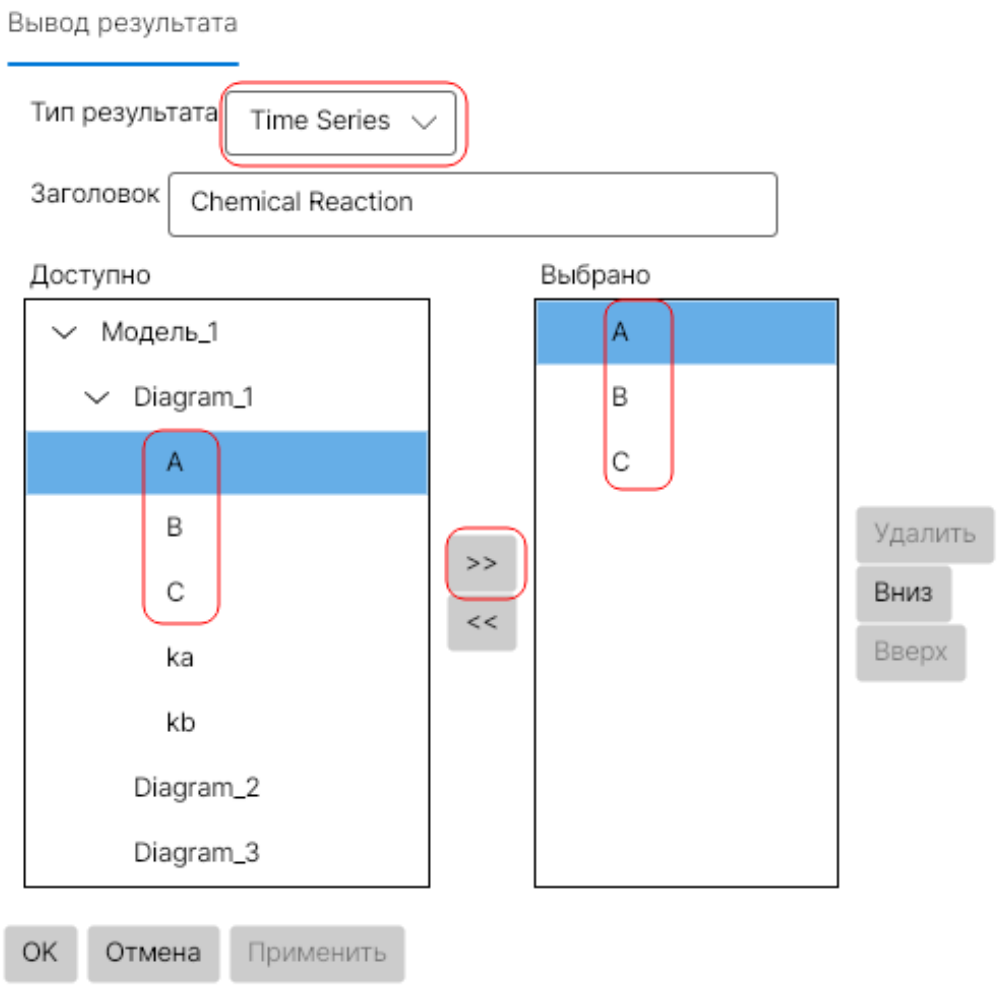

Рис. 2.4: Добавьте переменные интегралов на график для отображения.

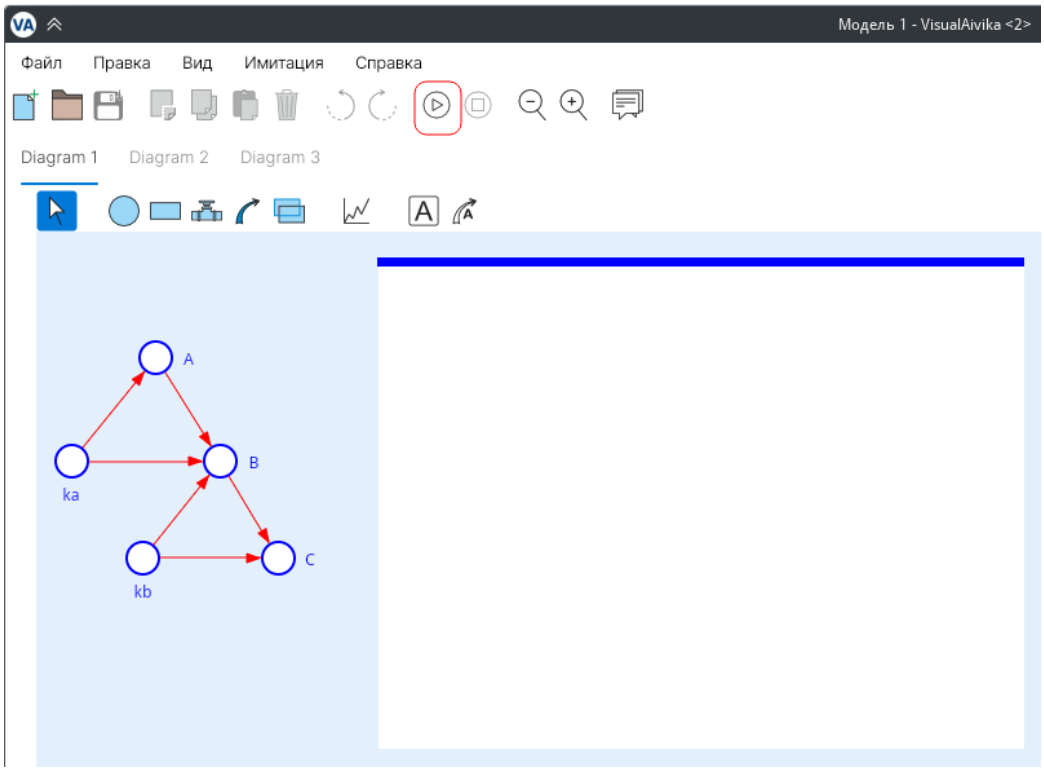

Рис. 2.5: Кнопка *Запуска имитации*.

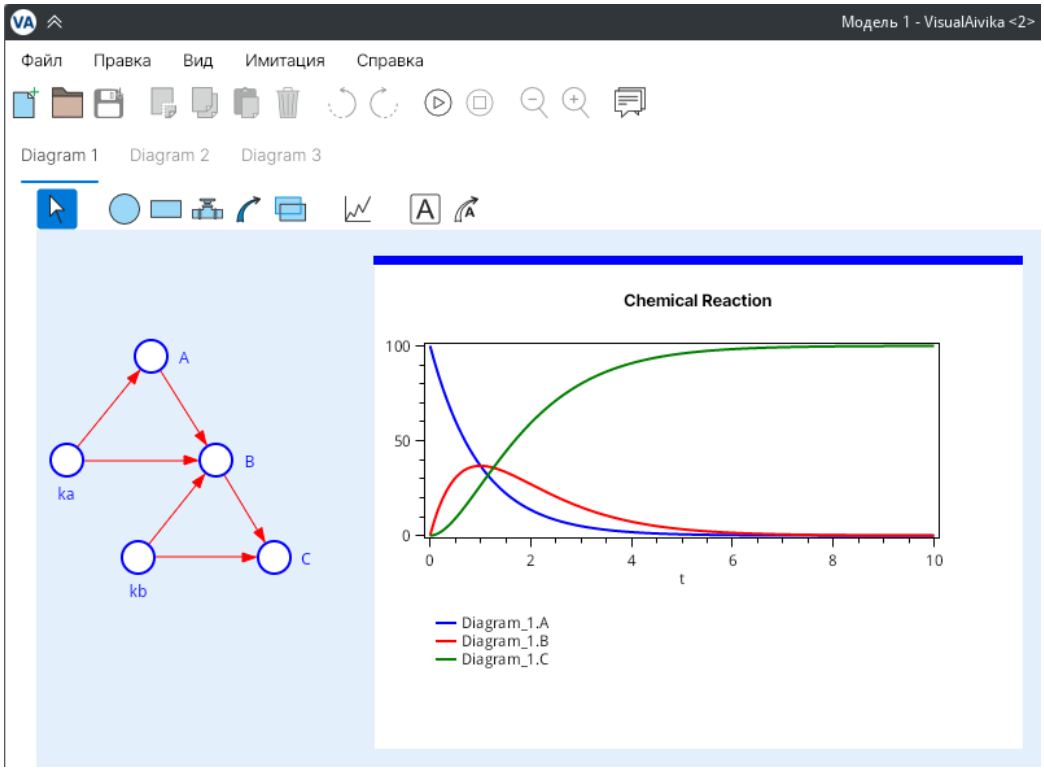

Рис. 2.6: Диаграмма после завершения имитации.

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-

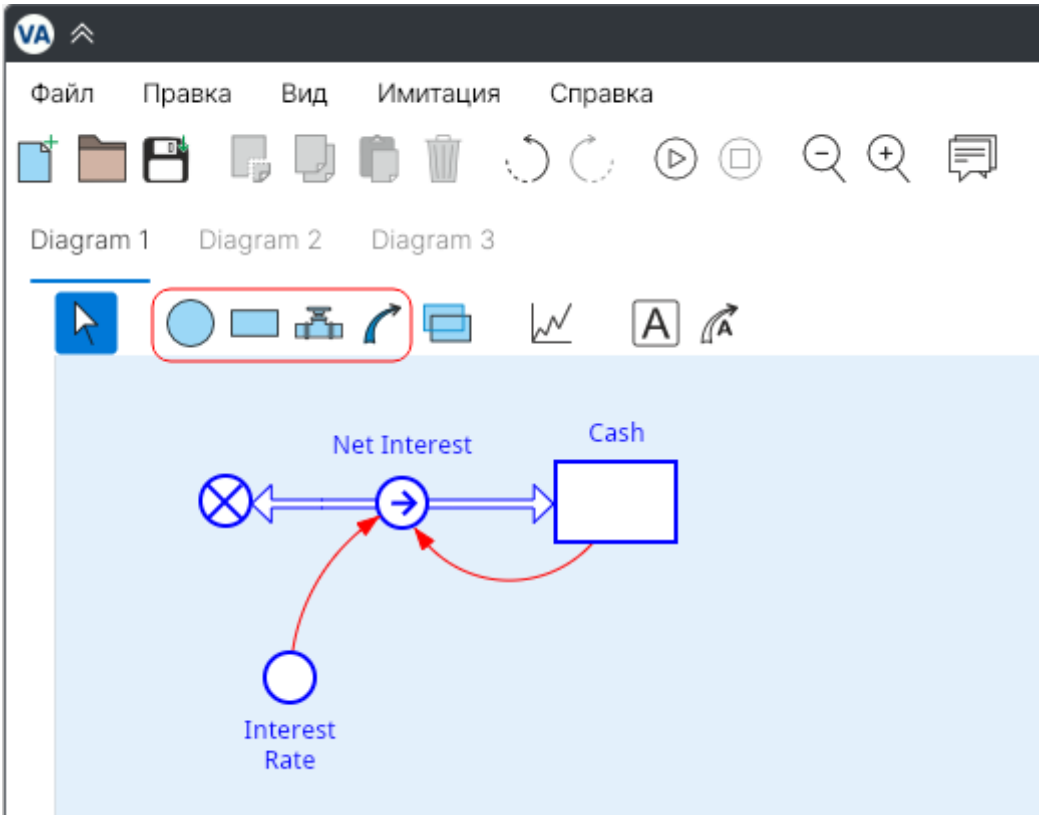

Рис. 3.1: Исходная потоковая диаграмма SFM для модели экспоненциального роста.

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 VisualAivika предоставляет разностороннюю поддержку уравнений, с помощью которых мы можем задавать и запускать достаточно сложные системы обыкновенных дифференциальных уравнений. Более того, VisualAivika имеет средства для проведения анализа чувствительности с помощью таких уравнений. В то же самое время VisualAivika позволяем нам моделировать и дискретно-событийные модели, которые могут быть скомбинированы с обыкновенными дифференциальными уравнениями.

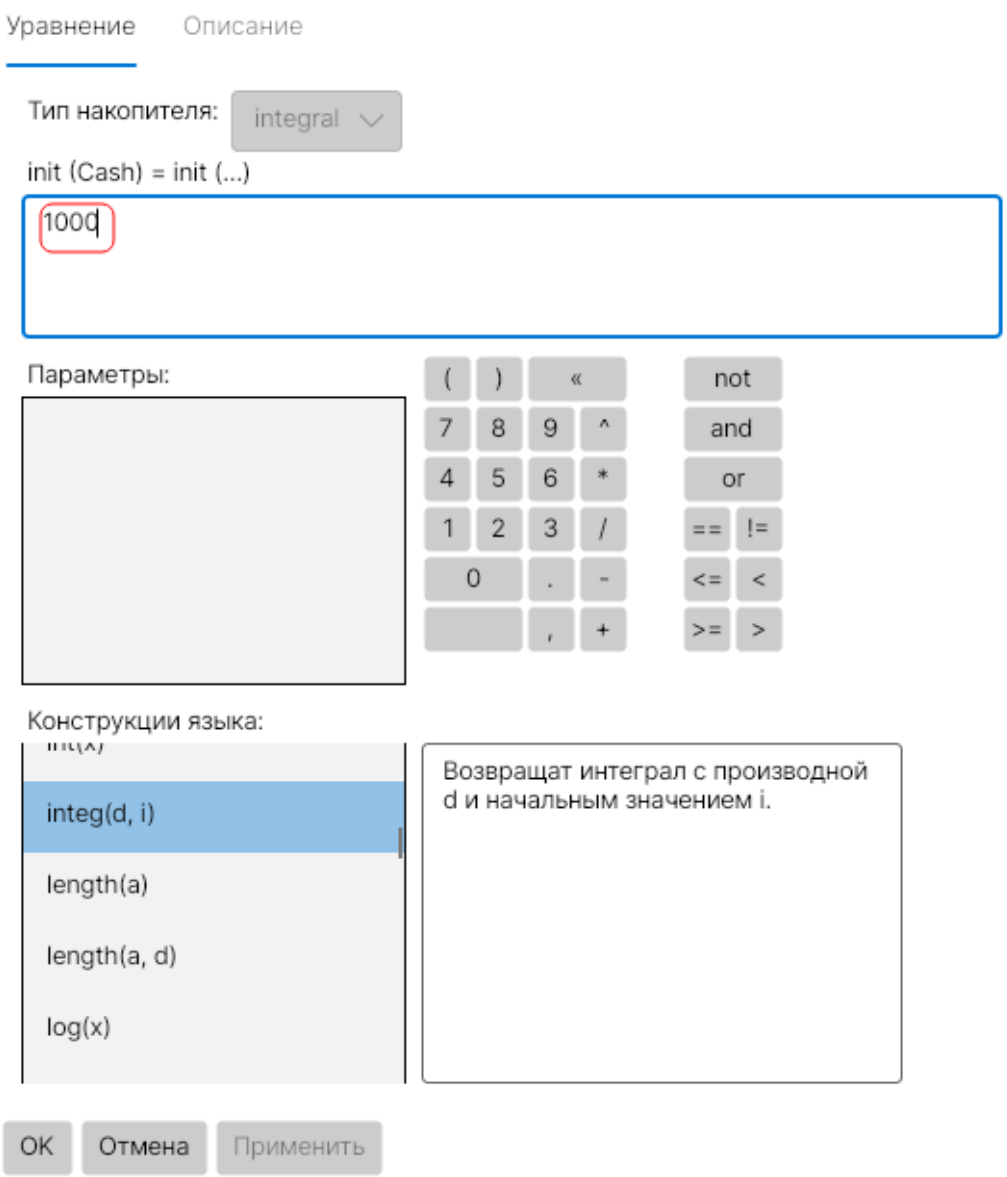

Рис. 3.2: Начальное значение накопителя SFM.

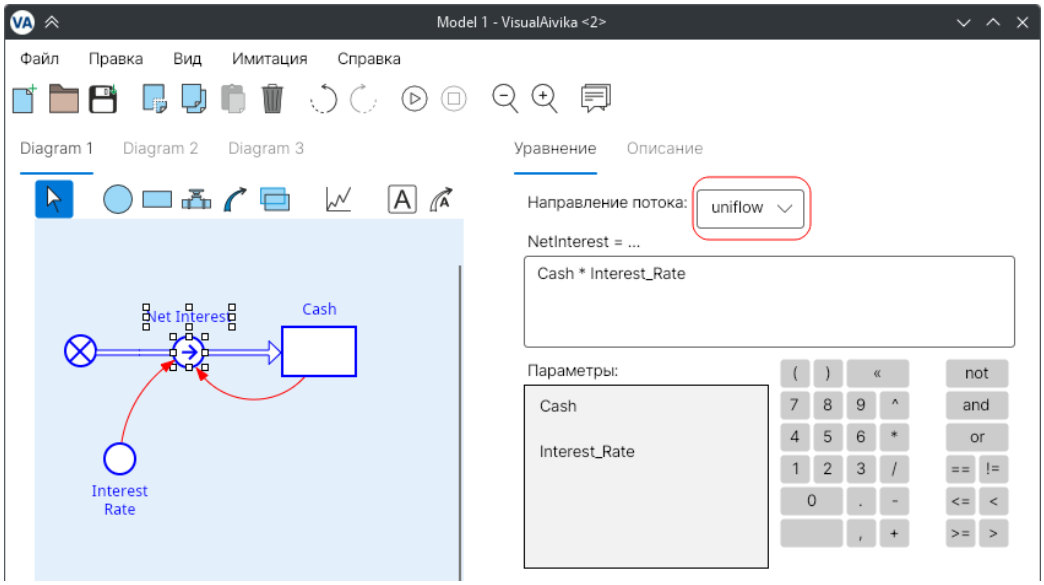

Рис. 3.3: Поток SFM становится однонаправленным.

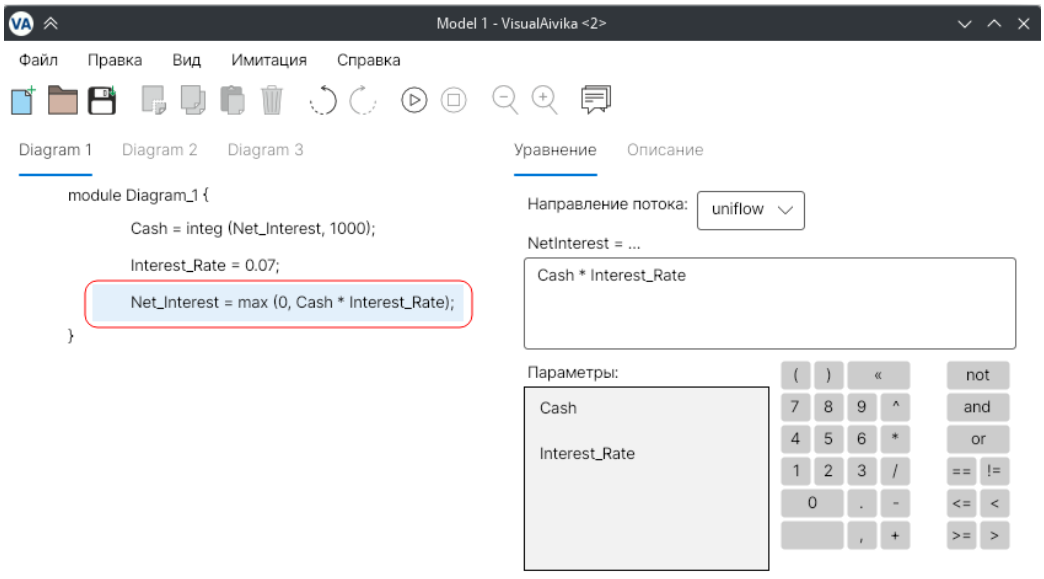

Рис. 3.4: Уравнения модели экспоненциального роста.

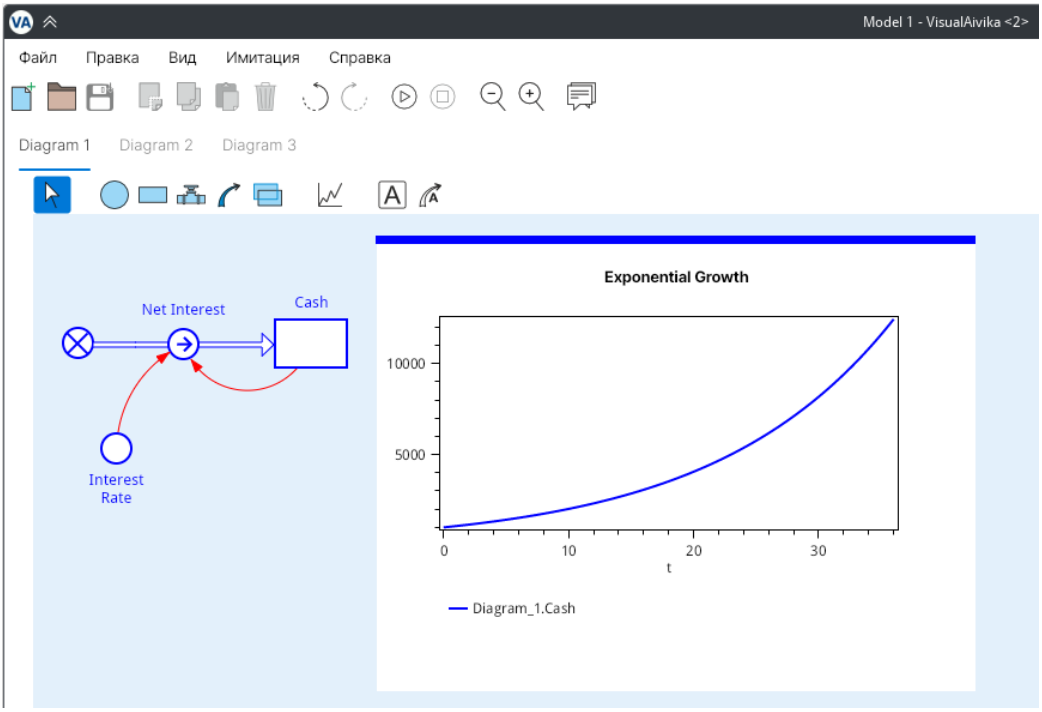

Рис. 3.5: График экспоненциального роста.

Глава 4

Как создать модель системы массового обслуживания

Для моделирования систем массового обслуживания VisualAivika использует подход похожий на язык моделирования GPSS[2], но реализация совершенная иная, которая имеет корни в функциональном программировании. Подобно GPSS система VisualAivika 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 VisualAivika 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.

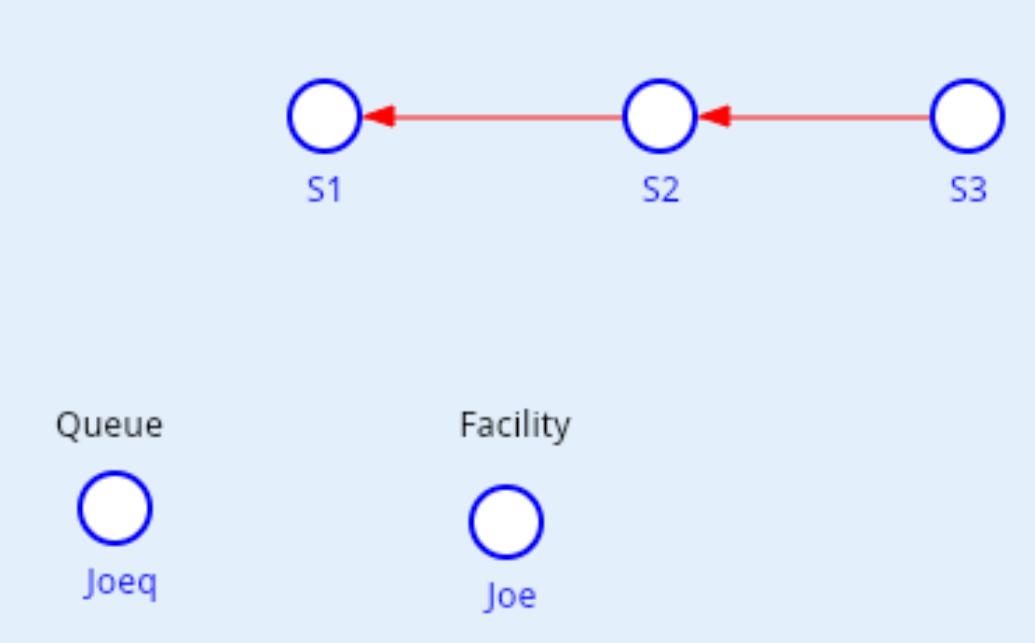

Рис. 4.1: Вычисления блоков как дополнительные элементы SFM.

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 VisualAivika выглядят так, как показано на рисунке 4.2. Здесь сразу заметим, что VisualAivika проверяет соответствие диаграммы и уравнений. Оранжевым цветом указаны предупреждения, которые можно проигнорировать, потому что мы не хотим усложнять диаграмму.

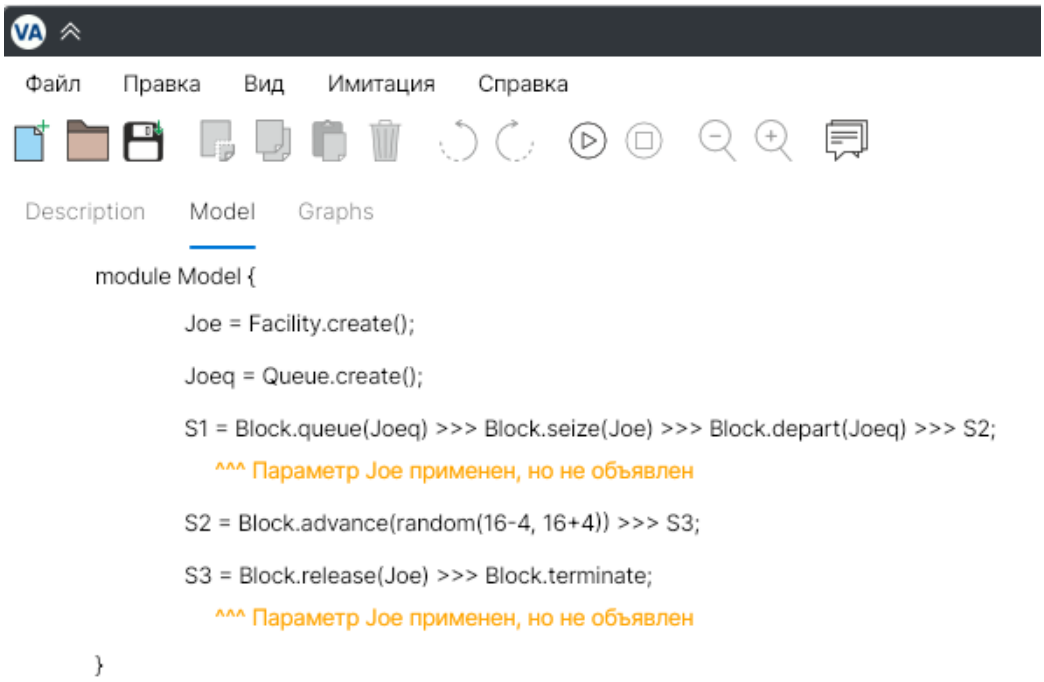

Рис. 4.2: Уравнения для вычислений блоков.

В VisualAivika 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-

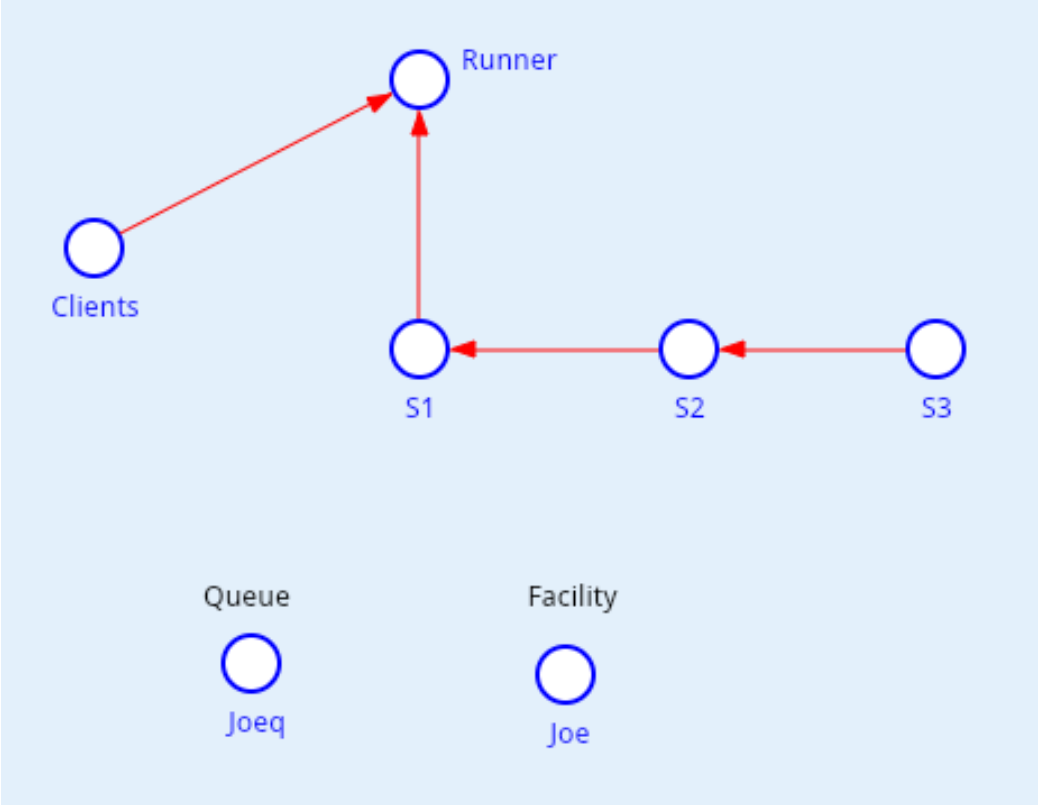

Рис. 4.3: Элементы завершенной модели.

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

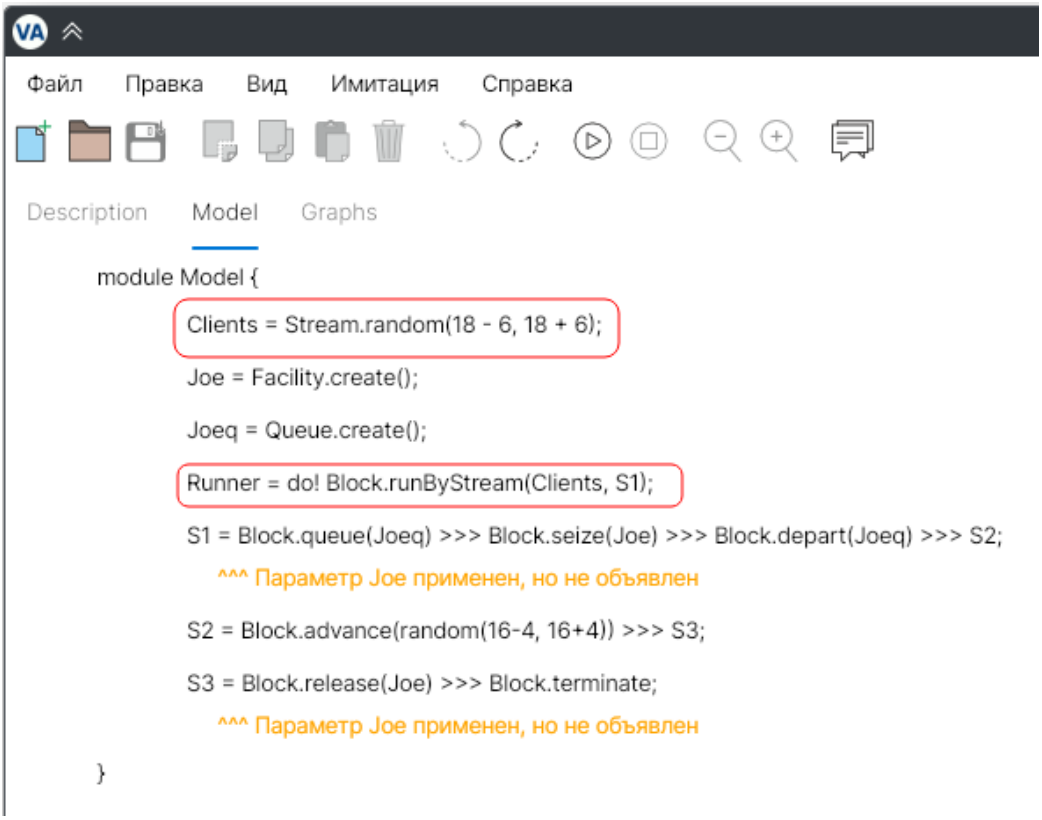

Рис. 4.4: Уравнения завершенной модели.

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

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

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

VisualAivika.

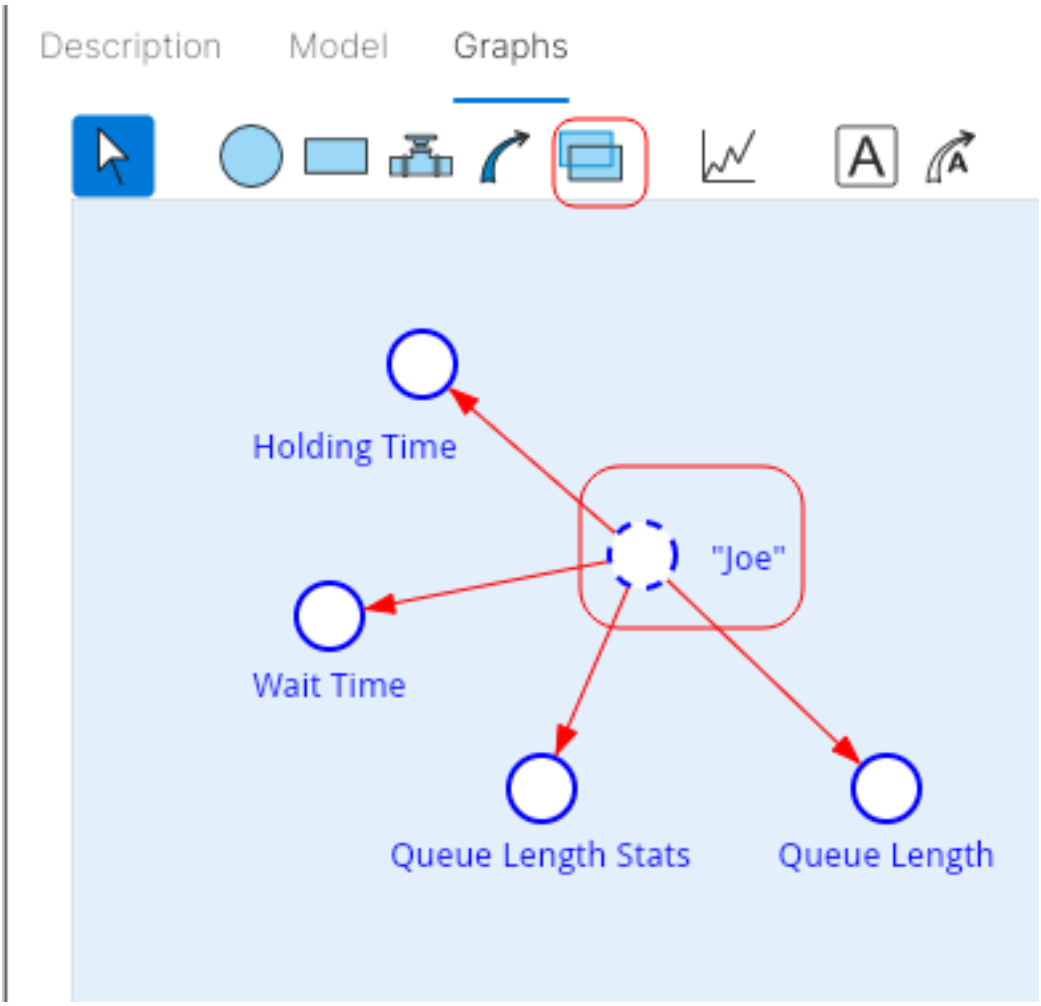

Рис. 4.5: Используйте ссылку SFM для уменьшения нагромождения на диаграммах.

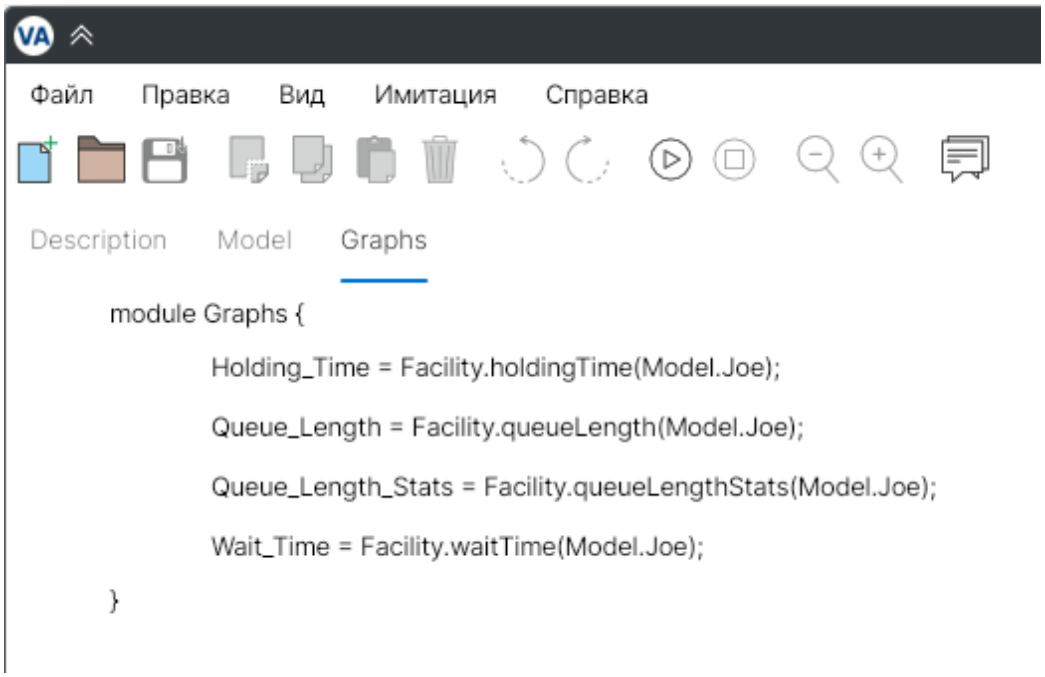

Рис. 4.6: Извлекаем свойства прибора.

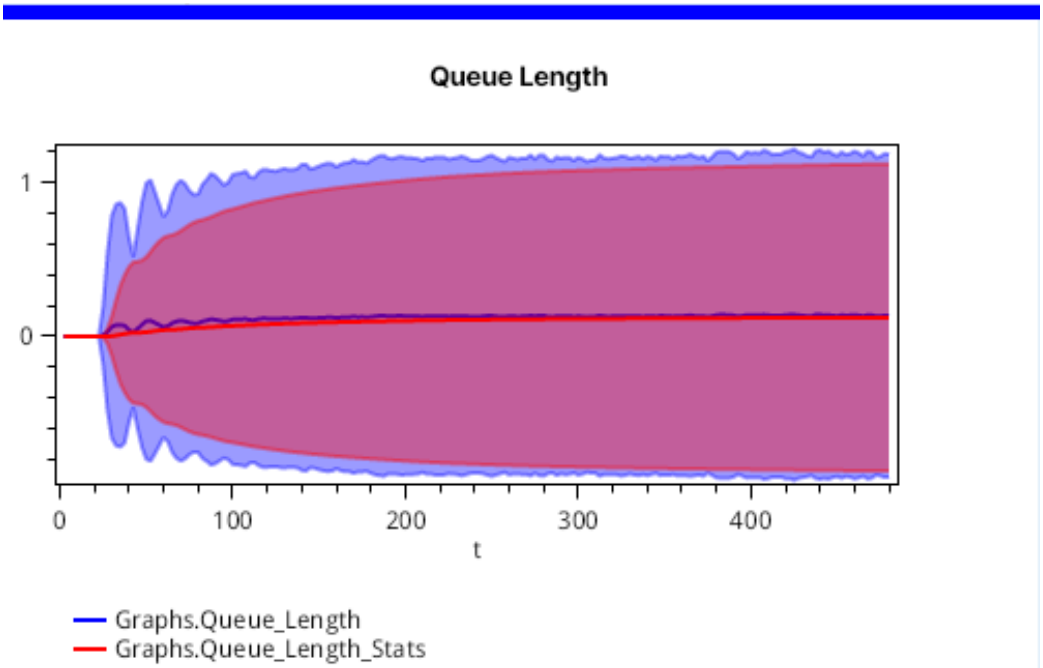

Рис. 4.7: Статистические результаты моделирования.

Глава 5

Язык моделирования

Язык моделирования VisualAivika 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`.

VisualAivika 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 VisualAivika.

- *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.
- *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.
- *Массив*. Массив из других переменных.

Название переменной в VisualAivika 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 VisualAivika 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.

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

Также VisualAivika 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

VisualAivika 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

VisualAivika 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;
```

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.

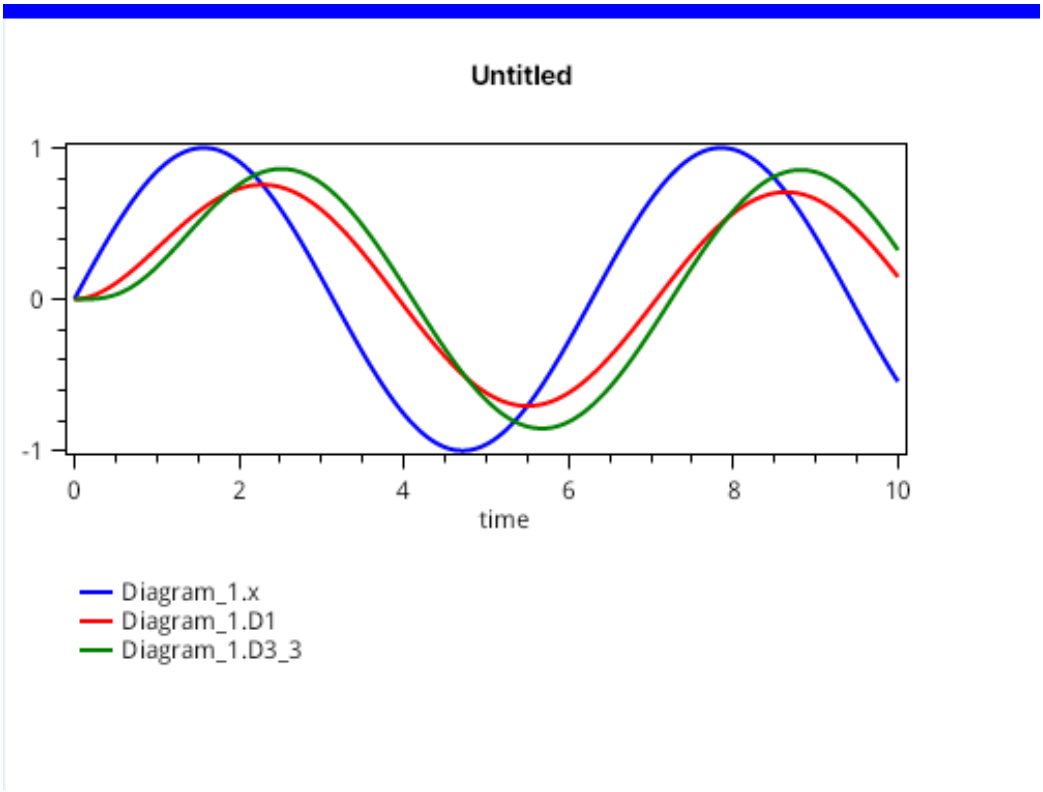

Рис. 5.1: Задержка первого порядка, D1, и задержка третьего порядка, D3_3, для заданного входа, x, и периода времени.

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 VisualAivika 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 VisualAivika имеет довольно нестандартные операторы `discrete` и `init`. Они связаны с самим движком моделирования в VisualAivika. 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 VisualAivika, 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:

$$[\ Element \mid Index <\text{-}\ Range\ ]$$

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:

$$[\ Element \mid Index1 <\text{-}\ Range1\ ,\ Index2 <\text{-}\ Range2\ ]$$

VisualAivika 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[3].

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.

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 VisualAivika 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*.

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, вероят-

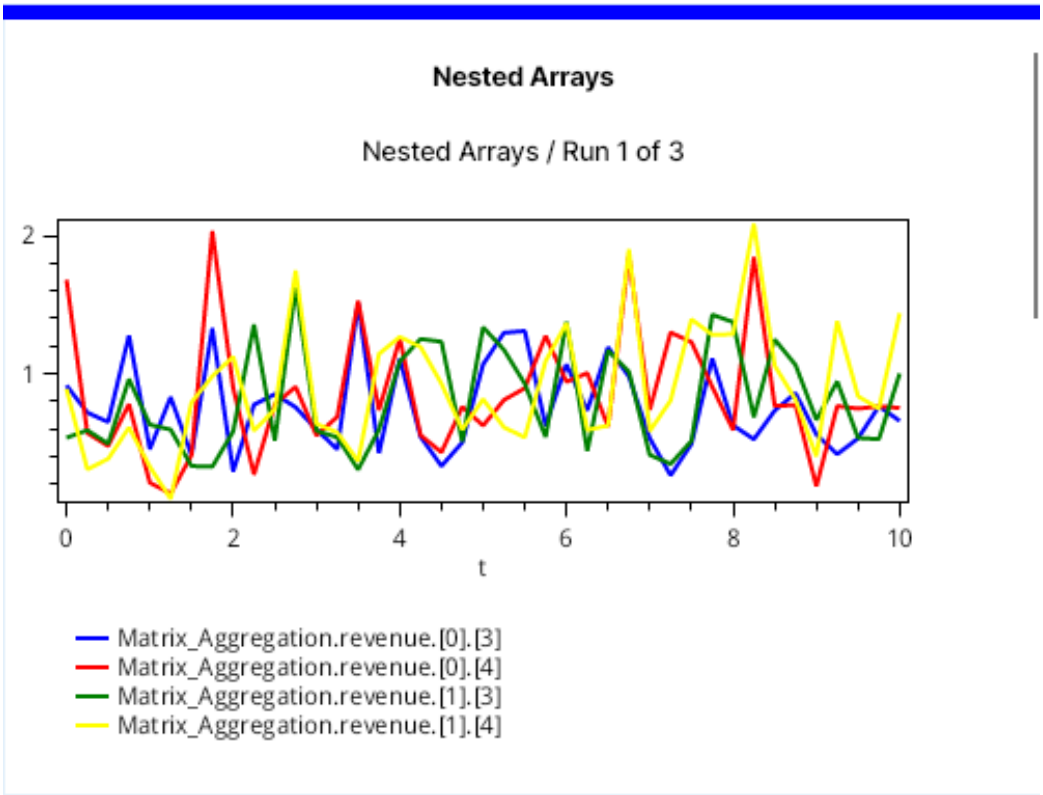

Рис. 5.2: При отображении массива на графике каждый элемент имеет свой индекс.

но, 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];
```

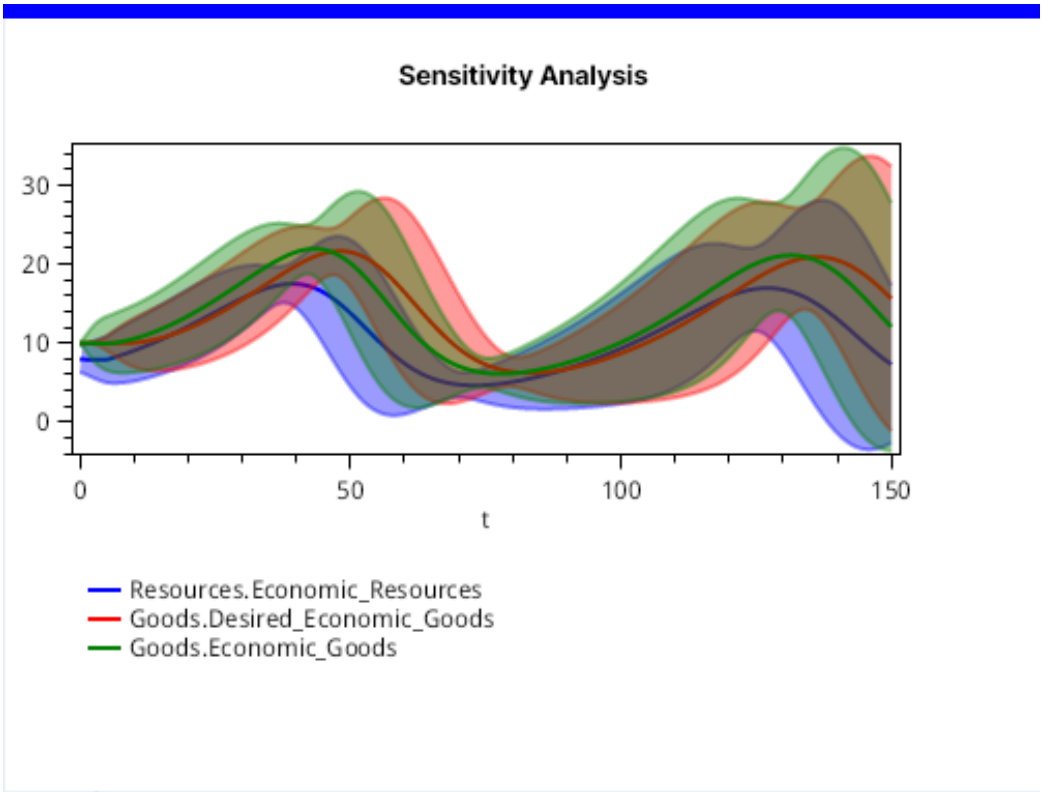

Рис. 5.3: График показывает тренды и доверительные интервалы, т.е. насколько устойчива модель относительно изменений внешних случайных параметров.

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 той же ассамблеи.

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 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.

Таблица 5.16: Блок генератора

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

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 транзакций с помощью этой цепочки блоков. Для этого нам нужен блок генератора, который может быть создан с помощью потока внешних событий.

Таблица 5.18: Запуск вычисления Block

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

Здесь последняя функция является сокращенной записью для вызова первой функции с помощью выражения, которое создает генератор через `Block.run(GeneratorBlock.byStream(s), b)`.

Обе 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 VisualAivika вводятся некоторые сущности для сбора статистики. Одной из таких сущностей является очередь. Транзакт може быть добавлен в очередь, а потом удален из нее. VisualAivika 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 VisualAivika, является прибор. Прибор — это такой ресурс, который может иметь только од-

ного 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.utilisationCountStats(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.

Example

```
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 A 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 Айвике. 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 устройство 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.contentStats(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.usedContent(s)`	Возвращает общее количество используемого содержимого для устройства `s` на текущий момент модельного времени
`Storage.contentUtilisation(s)`	Возвращает значение использования содержимого устройства для текущего модельного времени
`Storage.contentUtilisationStats(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 Вложенные модули

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

VisualAivika 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.

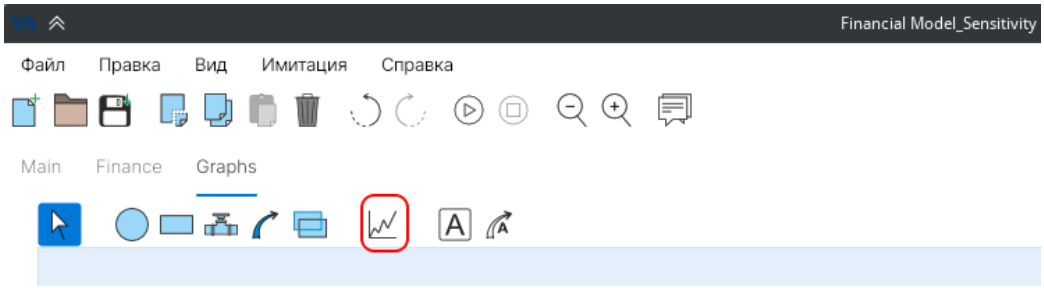

Рис. 6.1: Элемент для вывода результатов на панели диаграммы.

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 6.2. 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.

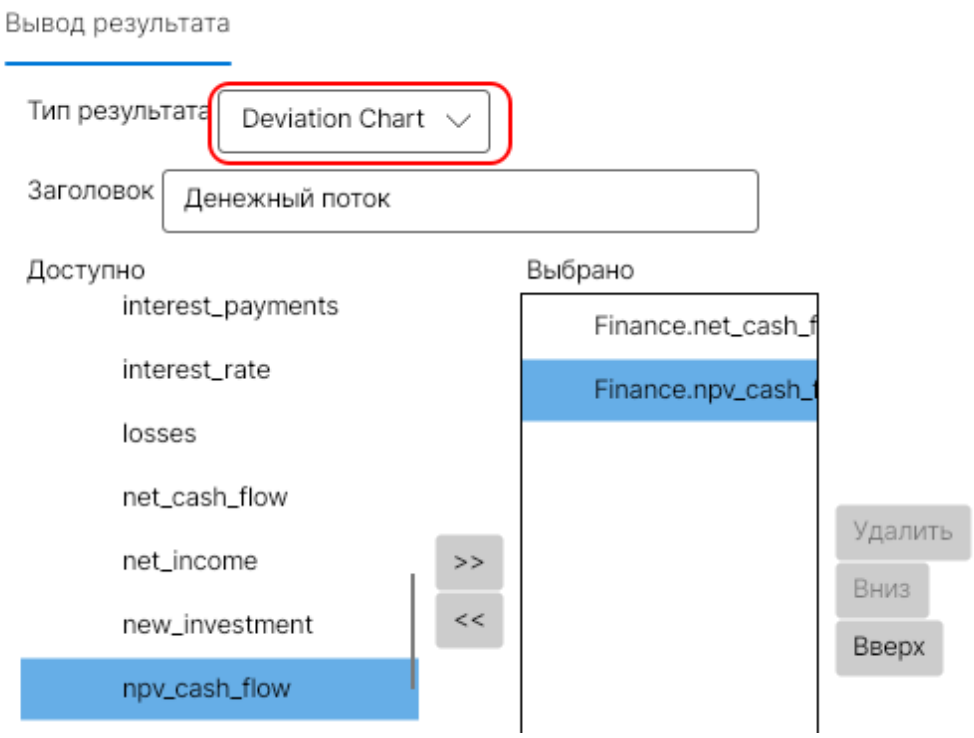

Рис. 6.2: Выбор представления для графика отклонения.

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

Представление графика отклонения может также показывать ста-

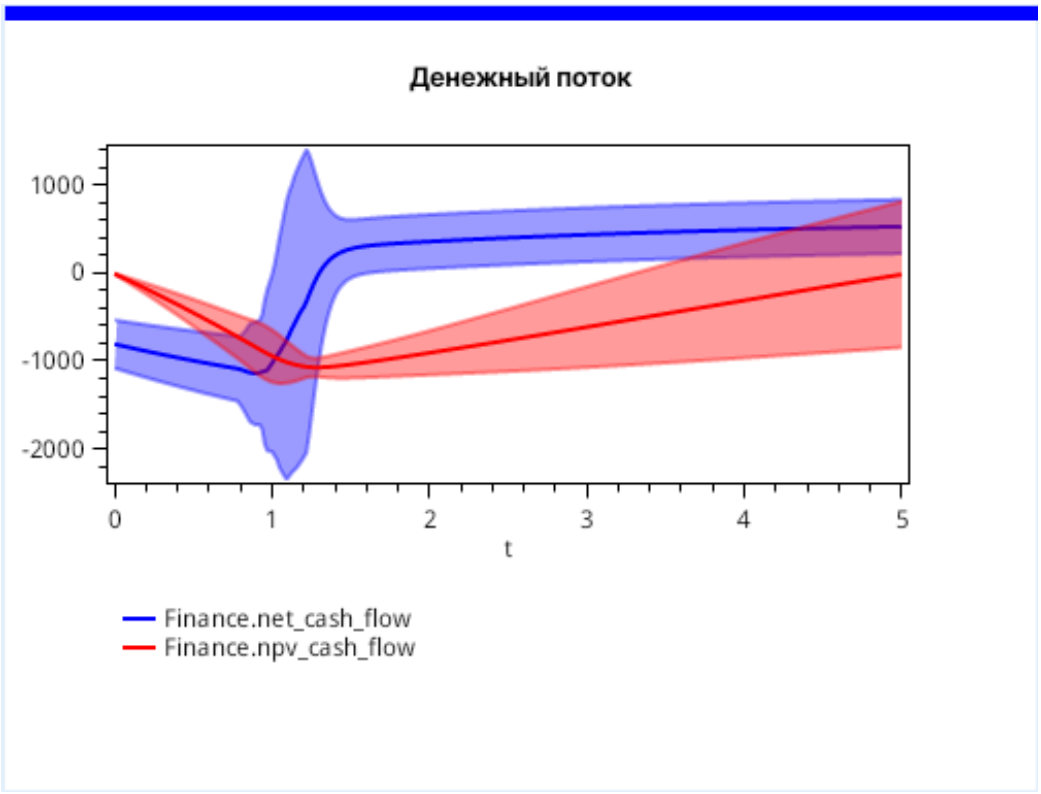

Рис. 6.3: Пример графика отклонения.

тические 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 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 еди-

ничного 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 6.4 показывает, как график временного ряда может выглядеть на диаграмме для единственного имитационного запуска.

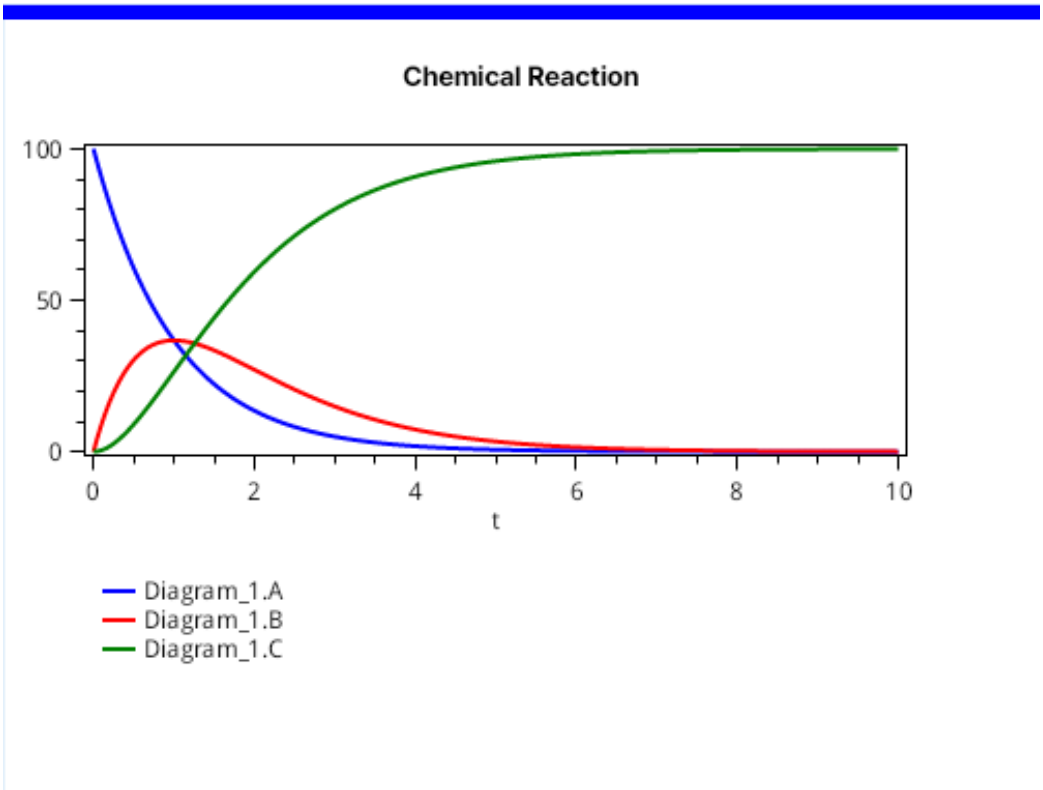

Рис. 6.4: Пример графика временного ряда.

6.3 График XY

График XY (*англ.* XY Chart) похож на график временного ряда, описанный в предыдущем разделе 6.2. 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 6.5 показывает, как график XY может выглядеть на диаграмме.

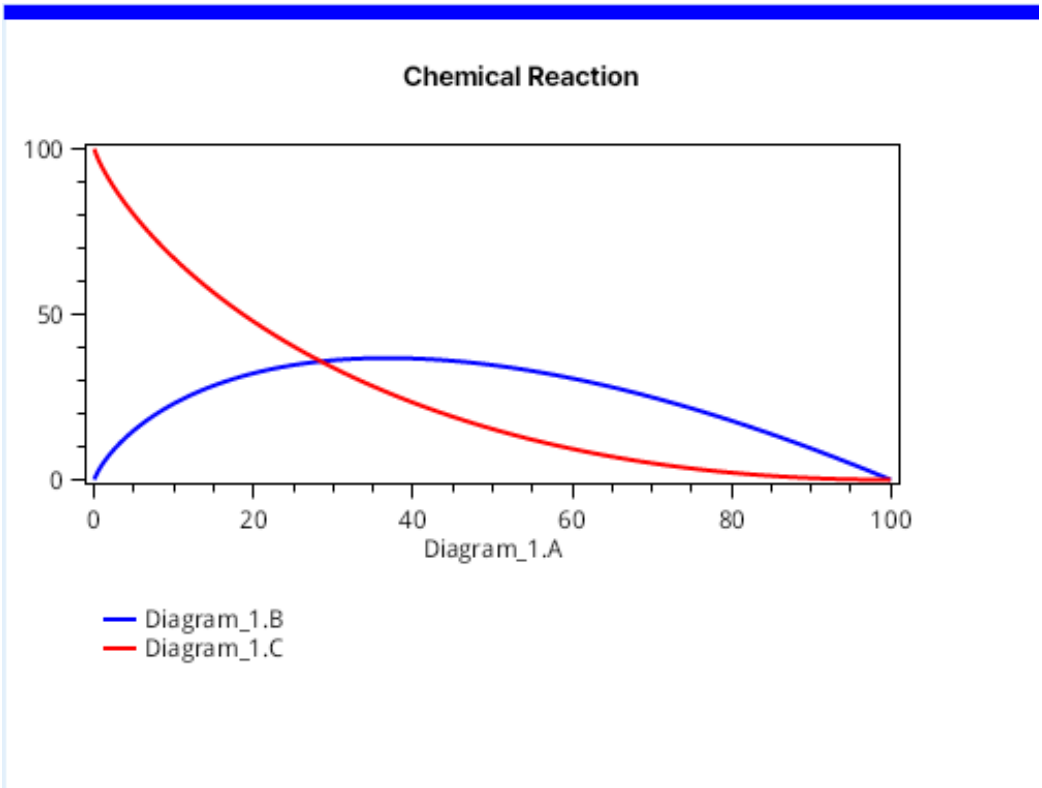

Рис. 6.5: Пример графика XY.

6.4 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 6.6 показывает, как таблица CSV может выглядеть на диаграмме.

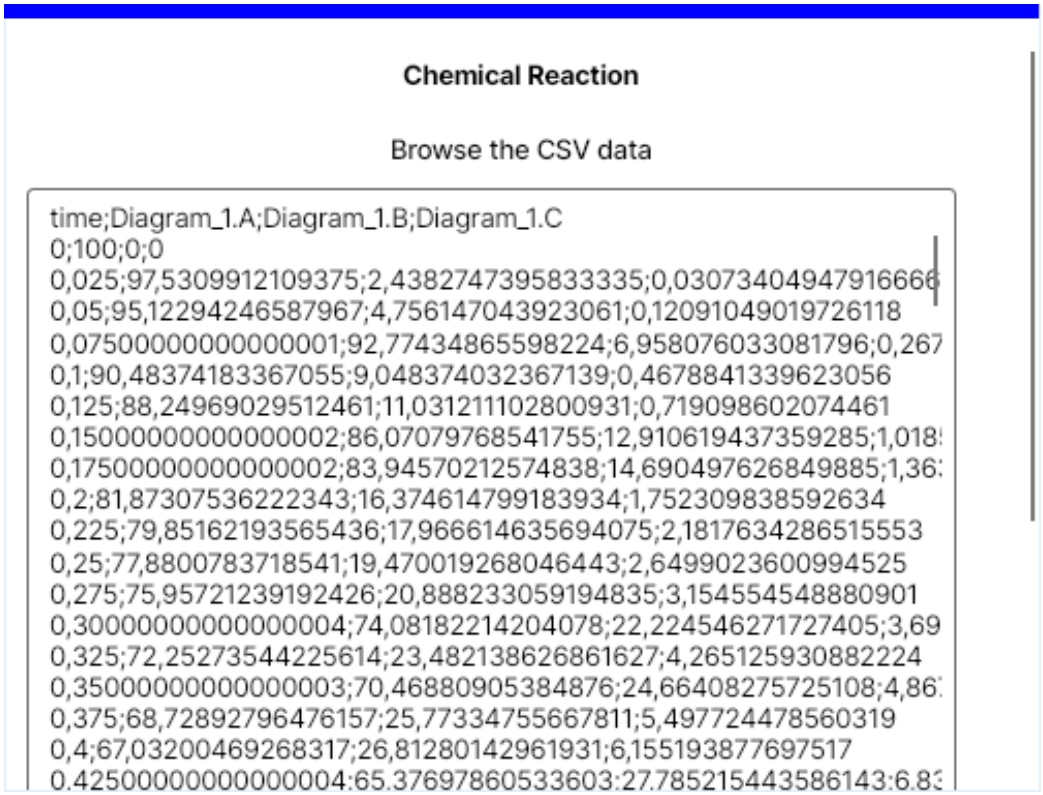

Рис. 6.6: Пример таблицы CSV.

6.5 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 6.7 показывает, как соответствующий элемент может выглядеть на диаграмме.

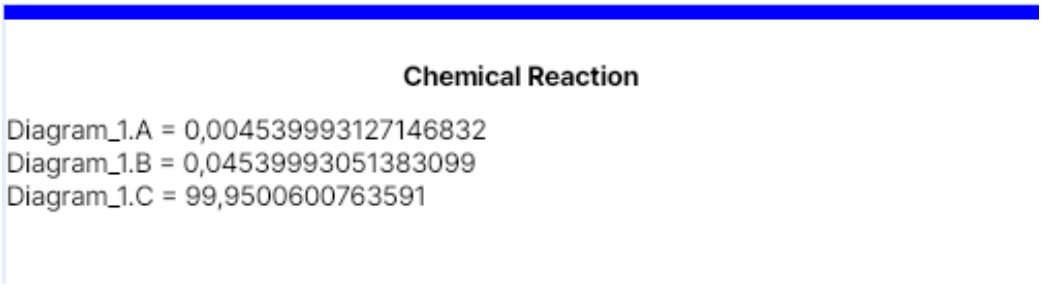

Рис. 6.7: Пример представления последних значений.

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.

6.6 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 6.8 показывает, как соответствующий элемент может выглядеть на диаграмме.

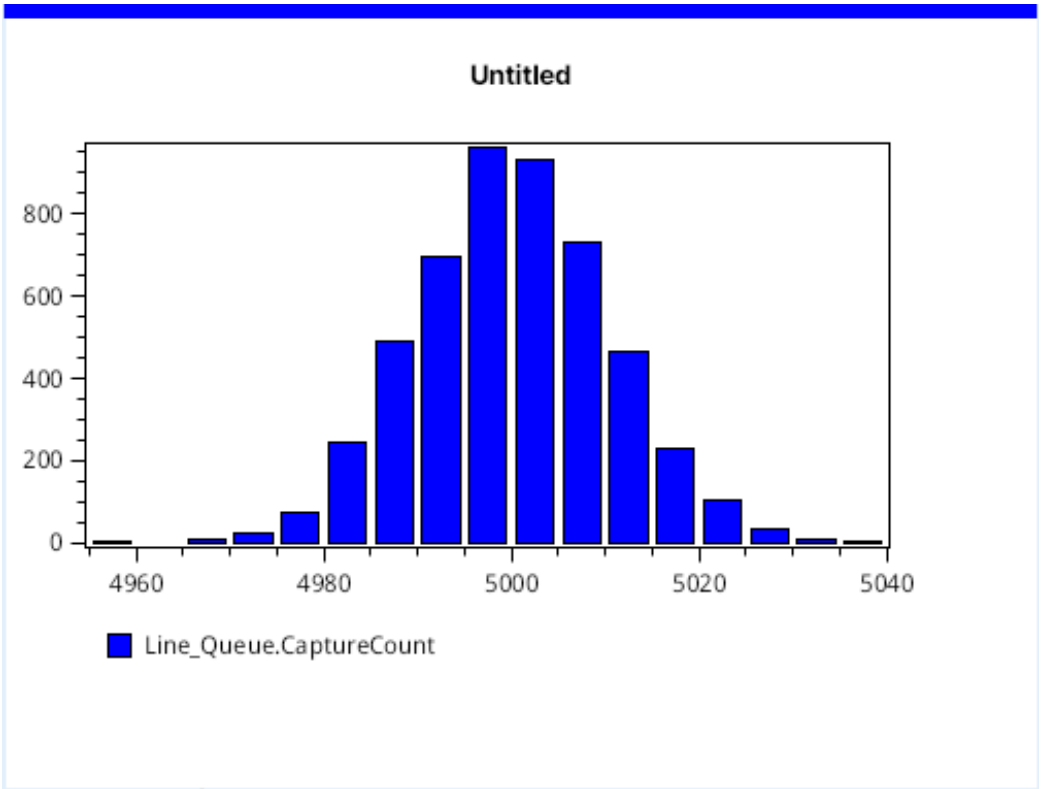

Рис. 6.8: Пример гистограммы последних значений.

6.7 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 6.9 показывает, как соответствующий элемент может выглядеть на диаграмме.

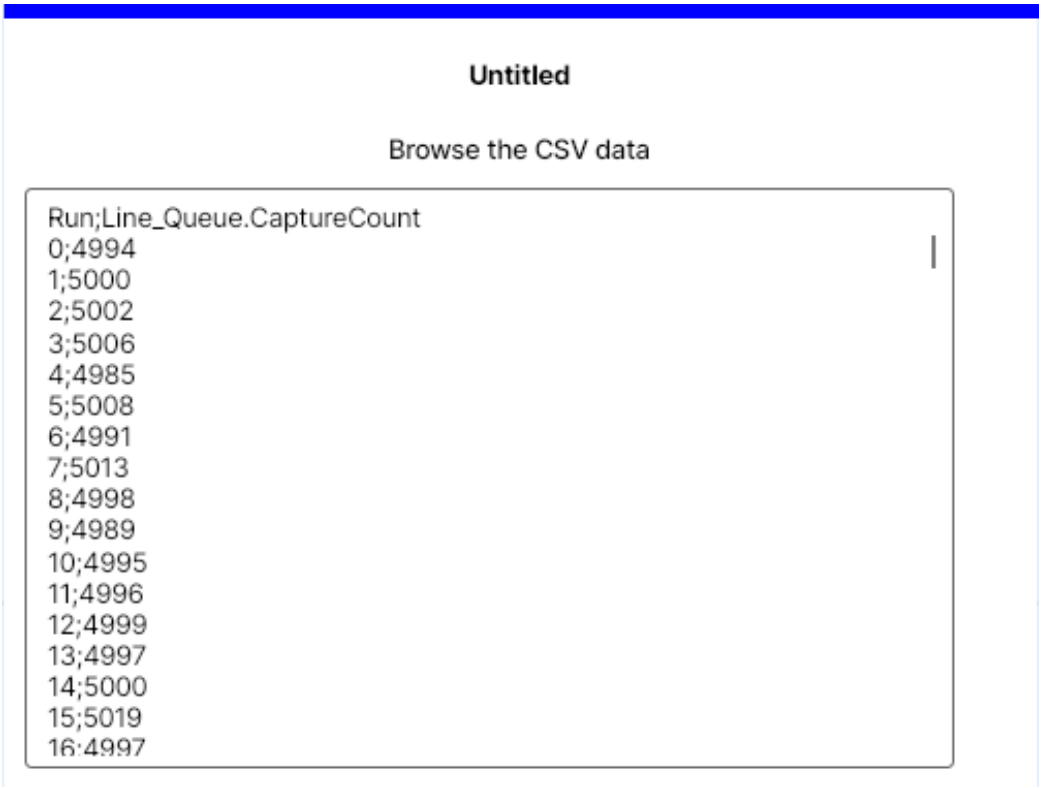

Рис. 6.9: Пример таблицы CSV последних значений.

6.8 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 6.10 показывает, как соответствующий элемент может выглядеть на диаграмме.

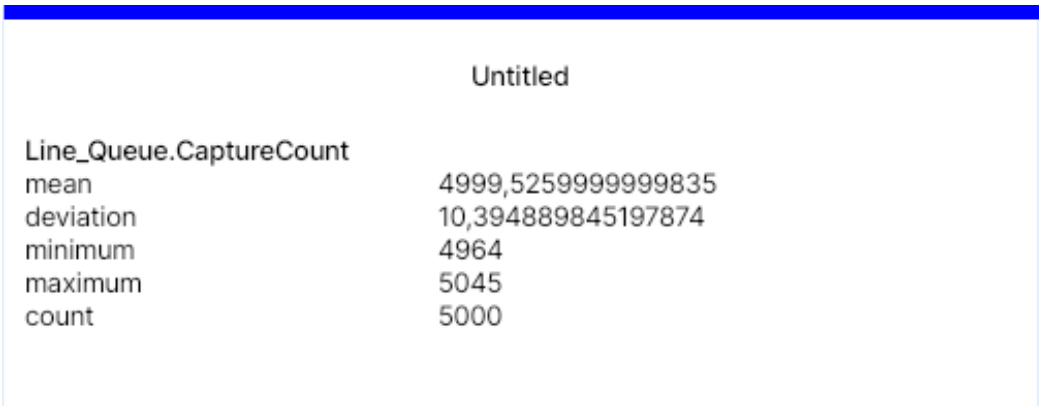

Рис. 6.10: Пример сводной статистики последних значений.

Есть один тонкий момент, свзязанный 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. Так что, этот элемент не показывает параметр `count` для временной статистики. В остальных случаях параметр `count` является тем, чем он и предполагается быть.

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 VisualAivika 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] Thomas Schriber. *Simulation using GPSS*. Wiley, 1974.

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