УДК 004.94

ВИЗУАЛЬНАЯ СРЕДА МОДЕЛИРОВАНИЯ VISUALAIVIKA

Д.Э. Сорокин (Йошкар-Ола)

Введение

В настоящей статье представлена авторская разработка VisualAivika. Это визуальная среда моделирования для системной динамики и дискретно-событийного моделирования. Документация и сама программа доступны по следующей ссылке [1]. Официально поддерживаются операционные системы Windows и Linux, хотя этот список может быть расширен[1]. Есть русская и английская версии интерфейса.

На момент написания статьи программа VisualAivika имела статус «бета». То есть, предварительная верификация пройдена, но еще требуется более полная и тщательная верификация.

Автором двигало желание, с одной стороны, импортозаместить продукты компаний, ушедших из нашей страны, а с другой – создать удобную в использовании и доступную для многих визуальную среду, где воплощены идеи, заложенные в более ранних работах автора по моделированию [2, 3]. Это продолжение данного направления исследований, только теперь уже в виде визуальной среды, где аудиторией могут быть люди, изучающие или практикующие моделирование, где не требуется ни навыков в программировании, ни, тем более, знаний в узкой области функционального программирования, которые требовались для использования более ранних результатов исследований. Здесь же в VisualAivika ничего из этого не требуется – автор попытался сделать среду моделирования простой, доступной и понятной.

Язык и визуальная среда моделирования VisualAivika

В программе используется свой собственный язык моделирования. Для системной динамики автор учился на примерах программ Vensim [4], iThink / Stella [5] и Berkley Madonna [6], тогда как для дискретно-событийного моделирования автор опирался на опыт создателей языка GPSS [7]. Причем, VisualAivika позволяет создавать комбинированные модели, где обыкновенные дифференциальные уравнения (ОДУ) можно сочетать с системами массового обслуживания в рамках одной имитационной модели.

На рис. 1 представлено изображение основного интерфейса программы для одной из версий VisualAivika.

[1] Автор запускал программу на компьютере фирмы Apple с операционной системой macOS.

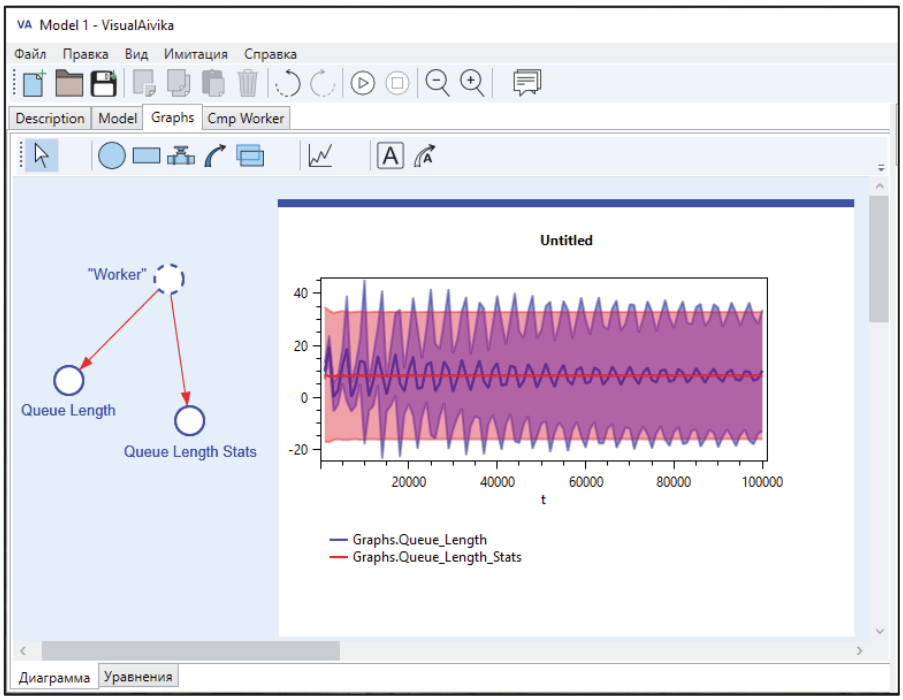

Рис. 1. Основное окно интерфейса системы моделирования VisualAivika

Здесь можно увидеть редактор диаграмм, где на диаграмме помимо сущностей модели присутствует также компонент для вывода результатов моделирования. В данном случае можно увидеть график отклонения, который для эксперимента по методу Монте-Карло показывает тренды и доверительные интервалы по методу «трех сигм».

Следует раскрыть часть истории создания программы VisualAivika. Та часть языка моделирования, которая используется для описания обыкновенных дифференциальных уравнений, основана на более ранних разработках и исследованиях, которые автор проводил совместно с доктором Захед Шейх (Dr. Zahed Sheikh). В свое время выпускалась и распространялась среда визуального моделирования Simtegra MapSys, программный код и алгоритмы которой полностью были разработаны автором данной статьи[1]. По обоюдному соглашению с доктором Захед Шейх автор настоящей статьи продолжил свои разработки и исследования в данной области уже в качестве независимого разработчика.

Что касается той части языка, которая относится к дискретно-событийному моделированию, то она уже была создана автором самостоятельно и независимо от других.

Системная динамика

В качестве иллюстрации языка системной динамики можно взять следующую систему обыкновенных дифференциальных уравнений из документации Berkeley Madonna [6]:

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

[1] Так было, как минимум, до 2011 года.

На языке VisualAivika эта система моделируется следующими уравнениями, как показано в листинге 1.

```
A = integ(-ka * A, 100);
B = integ(ka * A - kb * B, 0);
C = integ(kb * B, 0);
ka = 1;
kb = 1;
```

Листинг 1. Система ОДУ на языке VisualAivika

Здесь функция integ создает интеграл по заданной производной и заданному начальному значению. Уравнения могут быть заданы рекурсивно. Порядок уравнений не важен – решатель уравнений автоматически определяет зависимости.

К системам ОДУ сводятся модели потоков и накопителей (англ. Stock and Flow Maps), для которых есть специальная поддержка в редакторе диаграмм. На рис. 2 изображена часть такой модели, где есть и потоки, и накопители, и так называемые дополнительные элементы, соединенные стрелками, которые указывают на связи в модели.

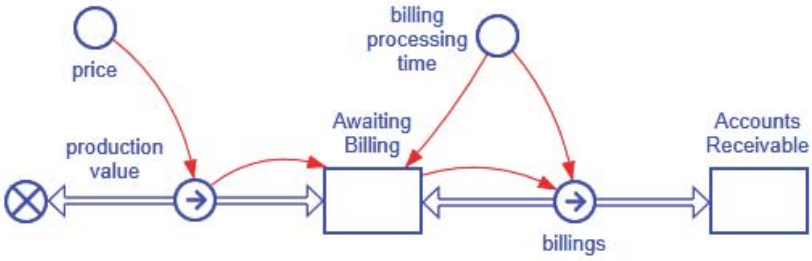

Рис. 2. Часть диаграммы потоков и накопителей в VisualAivika

Обычно модели должны соответствовать своим диаграммам, и VisualAivika предупреждает, когда возникает расхождение между диаграммой и моделью. Однако такие предупреждения не мешают запускать сами имитационные модели.

Поддержка массивов в системе VisualAivika

VisualAivika представлена в двух версиях: той, что использует интерпретацию моделей, и той, что использует компилятор уравнений. Наиболее важным отличием версии с компилятором является то, что она эффективнее и быстрее рассчитывает модели, где содержатся массивы. Массивы поддерживаются и там, и там, но в случае компилятора уравнений накладные расходы на поддержание массивов значительно ниже.

В качестве примера модели с массивами можно рассмотреть следующую модель из той же документации Berkeley Madonna, как показано в листинге 2.

```
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 ];
```

Листинг 2. Пример модели с массивами в VisualAivika

Здесь переменные C и M создают массивы размерности N+2 и N, соответственно. Форма записи похожа на то, что называется часто в программировании как «генератор списка» (англ. list comprehension). Только здесь будут массивы, а не списки. Причем массив M определяет интегралы.

Подобно Vensim в языке VisualAivika можно создавать промежуточные массивы внутри выражений, например, для агрегирования или для нахождения минимума и максимума.

В листинге 3 показаны уравнения для нахождения максимина и минимакса, соответственно.

```
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 ]);
```

Листинг 3. VisualAivika может создавать массивы внутри выражений

Версия с компилятором отличается тем, что здесь память под промежуточные массивы будет выделена только один раз в самом начале имитации, что призвано ускорить вычисления. Более того, индексирование по массиву почти не имеет накладных расходов в этой версии.

Однако преимуществом версии с интерпретатором является то, что ей не нужно, чтобы платформа .NET была установлена на компьютере пользователя (автор написал VisualAivika на языках C# и F#). Более того, версия с интерпретатором может работать и в браузере, не требуя предварительной установки программы. Только версия в браузере все же значительно медленнее своих десктопных версий, и браузерная версия пока существует экспериментально, и она нигде не была опубликована к моменту написания статьи.

Массивы могут быть созданы не только по интегралам и обычным функциям, но VisualAivika позволяет также создавать массивы очередей, ресурсов и вычислений блоков, которые будут обрабатывать транзакты[1].

Дискретно-событийное моделирование

При разработке этой части моделирующего языка автор опирался на свою собственную реализацию блоков [2], которая отдаленно напоминает блоки из языка моделирования GPSS.

Для иллюстрации основной идеи можно использовать следующую модель, написанную на языке GPSS, как показано в листинге 4 (модель 2A из книги [7]).

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

Листинг 4. Некоторая модель на языке GPSS

Аналогичная модель на языке VisualAivika записывается следующим образом, как показано в листинге 5.

```
S1 = Block.queue(Joeq) >>>
Block.seize(Joe) >>>
Block.depart(Joeq) >>> S2;
S2 = Block.advance(random(16-4, 16+4)) >>> S3;
S3 = Block.release(Joe) >>> Block.terminate;
```

Листинг 5. Аналогичная модель на языке VisualAivika

[1] На сайте VisualAivika [1] в разделе «Статьи» есть пример такой модели, которая называется «модель замкнутой сети очередей».

Блоки являются вычислениями, которые можно соединять через оператор композиции (>>>). Хотя на настоящий момент нет специальной визуальной поддержки, такие вычисления можно определять через дополнительные переменные диаграммы потоков и накопителей. Это будет не системная динамика, но визуально такие модели изображать сейчас можно.

Так, рис. 3 показывает, как может выглядеть такая модель на диаграмме. Через диаграммы потоков и накопителей можно определять и системы массового обслуживания, используя так называемые дополнительные переменные.

Рис. 3. Диаграмма потоков и накопителей

На рисунке порядок стрелок противоположен тому, как обрабатываются транзакты. В случае специальной поддержки в редакторе диаграмм это правило может измениться, но пока порядок такой.

Здесь можно сделать следующее теоретическое отступление.

Ранее автор в программной библиотеке Айвика [2], написанной на языке чистого функционального программирования Haskell, ввел вычисление Process для определения дискретных процессов. Там модели отдаленно напоминают то, как аналогичные модели задаются в языке моделирования SIMSCRIPT [8]. То есть, задается как бы линейный поток исполнения, где могут быть прерывания, задержки по времени, блокировки и тому подобное.

Затем в Айвике автор ввел вычисления Block для задания имитационных моделей аналогично тому, как это делается в языке моделирования GPSS. Использованные в VisualAivika и приведенные в листинге 5 уравнения и есть примеры вычисления Block.

Далее важно заметить, что Process является монадой, а Block – стрелкой Клейсли, построенной по этой монаде. Теория функционального программирования утверждает, что тогда Process и Block будут эквивалентными. То есть те модели, которые можно определить с помощью вычисления Process, можно также определить и с помощью вычисления Block, и наоборот.

Это вовсе не значит, что с помощью вычисления Process можно определить вообще все дискретные процессы, но все же можно ожидать, что вычисления блоков в VisualAivika являются достаточно репрезентативными. На этом теоретическое отступление можно закончить.

Вычисления блоков реализованы в VisualAivika так, что их можно сочетать с обыкновенными дифференциальными уравнениями в рамках единой комбинированной модели.

Более того, использование вычислений Block предполагает, что в будущем можно будет создавать вложенные подмодели аналогично тому, как GPSS STUDIO позволяет создавать и использовать ТЭБы [9]. Однако подобная функциональность в VisualAivika пока только в планах автора, о которых он писал ранее [3].

Композиция и запуск вычислений блоков

Благодаря использованию оператора композиции (>>>) цепочки блоков могут естественным образом сливаться в более крупные цепочки блоков. Более сложным является обратный случай разветвления блоков, когда обработка заявок из одного блока может распадаться на обработку в разных блоках, как показано в листинге 6.

```
P2 = Block.select(if Facility.isInterrupted(Prof) then Busy else P3);
```

```
P3 = Block.preempt(Prof, priorityMode=true, removalMode=false, transfer(Add)
via transact.P5) >>> P4;
P4 = …
Busy = Block.terminate;
```

Листинг 6. Разветвление блоков с последующим вытеснением транзакта на языке VisualAivika

Листинг 6 примерно соответствует следующему коду на языке моделирования GPSS, что показан в листинге 7 (пример 7-26 из книги [7]).

```
        GATE NI PROF,Busy
        PREEMPT PROF,PR,Add,5
        …
Busy    TERMINATE
```

Листинг 7. Аналогичное разветвление блоков с последующим вытеснением транзакта на языке GPSS

То есть, один поток обработки транзактов может распадаться на два в зависимости от того, выполнено некоторое условие или нет в рамках вычисления Block.select. Здесь в примере в случае невыполнения условия происходит возможное вытеснение транзакта в рамках вычисления Block.preempt, где заданному атрибуту транзакта (в приведенной модели VisualAivika это будет атрибут с названием “P5”) будет присвоено значение промежутка времени, которое еще транзакт должен был бы провести, будучи задержанным на имитации активности, но пока не был вытеснен.

Во многих вычислениях блоков в VisualAivika можно использовать значения атрибутов транзакта. Так, в продолжение примера из листинга 6 можно было бы продолжить имитацию активности после вытеснения, как показано в листинге 8.

```
Add = Block.advance (transact.P5 + 300) >>> LetGo;
LetGo = Block.release(Prof) >>> Block.terminate;
```

Листинг 8. Имитация активности через задержку с использованием атрибута транзакта на языке VisualAivika с последующим освобождением ресурса (прибора)

В VisualAivika реализованы следующие виды ресурсов: приборы, допускающие вытеснение единственного владельца, а также многоканальные устройства, которые может использовать множество транзактов одновременно. Подобно GPSS есть также сущность очереди для накопления статистики.

Создание и запуск транзактов в VisualAivika несколько отличается от GPSS. Сначала необходимо создать соответствующий поток внешних событий, затем создать блок генератора, а потом по такому генератору и заданной цепочке блоков запустить собственно саму обработку транзактов.

Для модели выше с вытеснением прибора запуск транзактов выглядит так, как показано в листинге 9. Только здесь блок генератора создается неявно в функции запуска.

```
Phone_Call_Stream = Stream.random(2000 - 500, 2000 + 500);
Phone_Call_Runner = do! Block.runByStream(Phone_Call_Stream, P1);
…
P1 = Block.priority(1) >>> P2;
```

Листинг 9. Пример создания и запуска транзактов в VisualAivika по заданному потоку событий и заданной цепочке блоков P1.

Здесь вычисление в цепочке блоков P1 устанавливает приоритет транзактов, а затем переходит к цепочке блоков P2 из листинга 6, который был приведен ранее.

Что касается потока внешних событий, то здесь в примере задается поток, где задержка между событиями имеет равномерное распределение в пределах 2000 ± 500.

Ансамбли транзактов

Каждый транзакт, созданный через подобный блок генератора (здесь генератор создается неявно в функции запуска), наделяется собственным ансамблем транзактов. Затем VisualAivika позволяет разделять транзакты на множество, где каждый транзакт из такого множества будет принадлежать тому же самому ансамблю. Транзакты из одного и того же ансамбля можно скомпоновать в один транзакт, а можно еще дождаться того момента, когда такие транзакты соберутся все вместе. Такое поведение похоже на то, как работают блоки ASSEMBLE и GATHER из языка моделирования GPSS.

Листинг 10 ниже показывает один из примеров использования (соответствует фрагменту модели 7-31 из [7]).

```
S1 = …
…
S4 = Block.split(1, S1) >>> S5;
…
S7 = Block.gather(24) >>> S8;
…
S11 = Block.assemble(12) >>> S12;
S12 = Block.terminate;
```

Листинг 10. Разбиение транзактов с последующей сборкой и компоновкой на языке VisualAivika

Здесь названия переменных подобраны специально так, чтобы указанные уравнения шли последовательно в панели уравнений VisualAivika, будучи отсортированными по названию.

Сочетание системной динамики и систем массового обслуживания

Что касается комбинирования систем ОДУ и дискретно-событийной модели, то в VisualAivika оно может происходить следующим образом.

Там, где вычисление блока ожидает выражение, где, кстати, можно использовать и атрибуты транзактов, там можно ссылаться на значения интегралов и других функций. Это связь от системы ОДУ в систему массового обслуживания.

С другой стороны, у приборов, многоканальных устройств и очередей можно запросить текущее состояние их свойств. Если такое свойство используется в системе ОДУ, то оно табулируется с заданным шагом, который равен шагу метода интегрирования. И при выводе таких свойств на графиках используются как раз табулированные значения. Это связь от системы массового обслуживания в сторону системы ОДУ.

Касательно таких функций, которые возвращают случайные числа, то в дискретно-событийной части такие величины вычисляются каждый раз заново при каждом использовании, а вот в системе ОДУ они табулируются.

Текущее модельное время везде имеет актуальное значение, как в дискретно-событийной части, так и в системе ОДУ.

Сбор статистики и эксперимент по методу Монте-Карло

Еще стоит заметить, что некоторые свойства ресурсов представляют собой статистическую сводку: количество наблюдений, среднее, отклонение, минимум, максимум, могут быть и другие атрибуты. Такие значения статистической сводки являются неизменяемыми. Более того, их можно отображать на графике отклонения. Например, на графике можно показать время ожидания в очереди.

Причем согласно [10] статистическая сводка может быть двух видов: статистика по наблюдениям (англ. statistics based upon observations) и статистика по времени (последнее название является авторским переводом для англоязычного термина time persistent variable statistics).

Так, для длины очереди можно определить статистику по времени, тогда как для времени ожидания можно посчитать статистику по наблюдениям.

При выводе графика отклонения (по правилу «трех сигм») автор делает допущение, что статистику по наблюдениям в каждой точке модельного времени можно объединять для разных имитационных запусков из одного эксперимента по методу Монте-Карло. Предполагается, что разные запуски дают одинаково-распределенные независимые величины. Поэтому среднее и дисперсия объединяются.

Причем, если взять статистику по времени, то она может быть сконвертирована в аналогичное значение статистики по наблюдениям, где только количество наблюдений теряет свой прежний смысл[1], а вот первые два момента сохраняют свои значения, то есть среднее и дисперсия остаются как есть. То же касается и экстремумов – информация о минимальном и максимальном значениях также сохраняется.

К примеру, на одном графике отклонения можно отобразить тренд и доверительные интервалы и для текущей длины очереди (которая табулируется, и этот эффект сильный), и для накопленной статистики по длине очереди (эффекта от табулирования почти нет).

Если процесс еще не стал стационарным, а время обработки распределено одинаково, то часто будет видно, как графики не сходятся. Тогда как для стационарного процесса обычно такие графики сходятся. Более того, длина очереди должна стабилизироваться.

Рис. 4 иллюстрирует, что процесс не стал еще стационарным, где длина очереди и статистика по ней не сошлись для метода Монте-Карло для заданного временного промежутка (пример 2B из [7]) при одинаково распределенном времени обработки. На коротком промежутке времени случайный процесс не успевает стабилизироваться, о чем свидетельствует возрастающий тренд для длины очереди.

[1] Такое значение статистики называется в программе «нормализованным», когда значение статистики по времени конвертируется в значение статистики по наблюдениям.

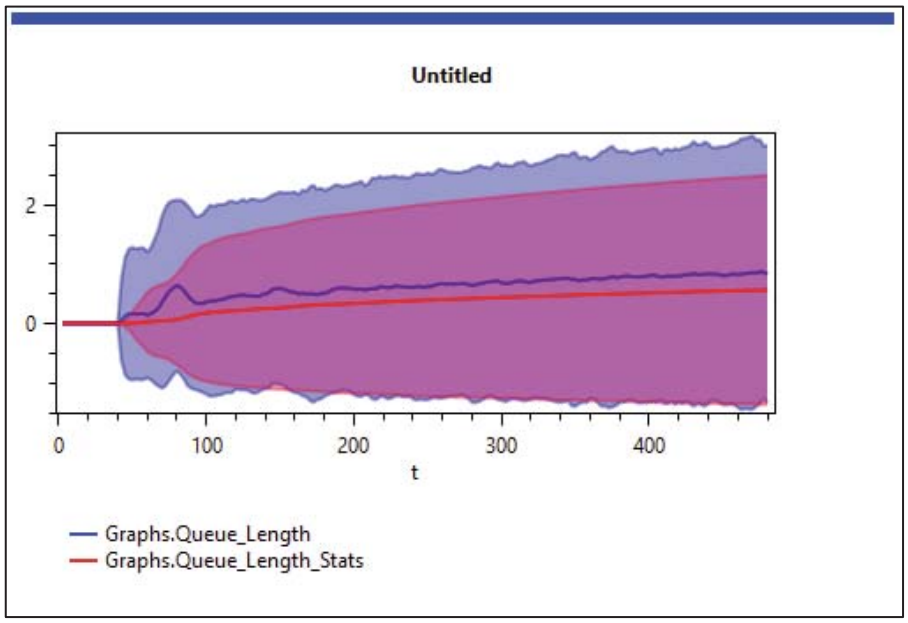

Рис. 4. Случайный процесс на коротком промежутке времени

Более того, видно, что тренд для длины очереди растет и не успевает стабилизироваться. Однако если в этой модели увеличить конечное время моделирования на два порядка, то оба графика примерно сойдутся.

К сожалению, в VisualAivika пока нет сброса статистики для ресурсов. Так может сложиться, что случайный процесс уже стабилизировался, а статистические показатели все еще будут сильно зависеть от ранних значений. Поэтому сброс статистики для ресурсов – один из приоритетов для будущих версий VisualAivika.

Обсуждение

Если сравнивать VisualAivika с другими программными средствами, то поддержка более одной парадигмы моделирования не является чем-то новым. Так, поддержка системной динамики и дискретно-событийного моделирования реализована в программной системе AnyLogic [11], которая к тому же поддерживает еще и агентное моделирование.

В целом языки моделирования для системной динамики в основе своей более или менее похожи. Различия начинаются, когда речь заходит о поддержке массивов или макросов. Могут отличаться и набором дополнительных функций, а также некоторыми дополнениями, сближающими где-то такие системы с дискретно-событийным моделированием, как, например, в iThink есть конвейеры наряду с интегралами для определения накопителей.

Однако в случае VisualAivika речь идет о том, что вводится также полноправный язык моделирования непосредственно для имитации систем массового обслуживания с ресурсами, транзактами и очередями. На взгляд автора, получился язык, который на уровне идей близок к используемому в GPSS STUDIO [9] языку GPSS, хотя и реализация совершенно другая, и она основана на концепциях функционального программирования. Именно благодаря тому, что блоки рассматриваются как вычисления, их удалось также трактовать в программе тем же способом, каким задаются системы ОДУ – просто еще другие типы данных в уравнениях.

Вычисления блоков обладают свойством композиционности. Это когда такие вычисления можно комбинировать друг с другом для создания новых вычислений.

Это – основная отличительная особенность, которую дает использование приемов и методов функционального программирования. Можно даже сказать, что в этом заключается сама суть функционального программирования – в возможности создавать композицию от простого к более сложному. И автор пока не может сказать, насколько такой подход является уникальным для визуальных сред моделирования.

Наиболее полно этот потенциал может раскрыться при добавлении возможности создания вложенных подмоделей. На уровне теории тут выглядит просто (ранее было реализовано в [2, 3]), если только ограничиться написанием программного кода. Однако удобная в использовании реализация для визуальной среды требует времени и затрат.

Заключение

Как было указано в начале статьи, VisualAivika является авторским проектом, осуществляемым на собственные средства. Это визуальная среда моделирования для решения задач системной динамики и дискретно-событийного моделирования. Пока проект находится в стадии более тщательного тестирования и верификации.

Литература

1. VisualAivika. URL: https://visualaivika.ru/ (дата обращения: 09.09.2025).
2. **Сорокин Д.Э.** Распределенное имитационное моделирование с Aivika // Прикладная информатика. 2019. Т. 14. № 4 (82). С. 73–89. DOI: 10.24411/1993-8314-2019-10028.
3. **Сорокин Д.Э.** DVCompute++ Simulator: декомпозиция для дискретно-событийного моделирования // Прикладная информатика. 2023. Т. 18. № 3. С. 72–91. DOI: 10.37791/2687-0649-2023-18-3-72-91.
4. Ventana Systems. URL: https://vensim.com (дата обращения: 14.05.2025)
5. isee systems. URL: https://www.iseesystems.com/ (дата обращения: 14.05.2025)
6. **Berkeley Madonna.** URL: http://www.berkeleymadonna.com (дата обращения: 14.05.2025)
7. **Шрайбер Т.Дж.** Моделирование на GPSS / пер. с англ. – М.: Машиностроение, 1980. – 592 с.
8. **Stephen V. Rice, Harry M. Markowitz, Ana Marjanski and Stephen M. Bailey.** The SIMSCRIPT III programming language for modular object-oriented simulation // Proceedings of the 37th Winter Simulation Conference, Orlando, FL, USA, December 4-7, 2005, pp. 621-630, DOI: 10.1109/WSC.2005.1574302
9. **Девятков В.В., Девятков Т.В., Федотов М.В.** Имитационные исследования в среде моделирования GPSS STUDIO / под ред. В.В. Девяткова. – М.: Вузовский учебник: ИНФРА-М, 2018. – 284 с
10. **A. Alan Pritsker and Jean J. O'Reilly.** Simulation with Visual SLAM and AweSim (2nd. ed.). John Wiley & Sons, Inc., USA, 1999. - 828 p.
11. **Карпов Ю.Г.** Имитационное моделирование систем. Введение в моделирование с AnyLogic 5. – СПб.: БХВ–Петербург, 2005. – 400 с.