

Лабораторная работа № 1
ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ РАБОТЫ СЕРВЕРА

ЦЕЛЬ РАБОТЫ: научиться использовать средства имитационного моделирования СМО на примере подбора эффективных параметров работы сервера.

ПОСТАНОВКА ЗАДАЧИ:

Сервер обрабатывает запросы, поступающие с автоматизированных рабочих мест с интервалами, распределенными по экспоненциальному закону со средним значением IntPostZ мин. Время обработки поступающих запросов зависит от производительности сервера Q (оп/с) и вычислительной сложности запросов, распределенной по нормальному закону с математическим ожиданием $S1$ (оп) и среднеквадратическим отклонением $S2$ (оп). Сервер имеет входной буфер ёмкостью emkBuf запросов. В случае полной занятости входного буфера поступающий запрос теряется. Время моделирования 1 ч.

Сервер представляет собой однофазную систему массового обслуживания разомкнутого типа с ограниченной входной ёмкостью, то есть с отказами, и абсолютной надёжностью (рис. 7.1).

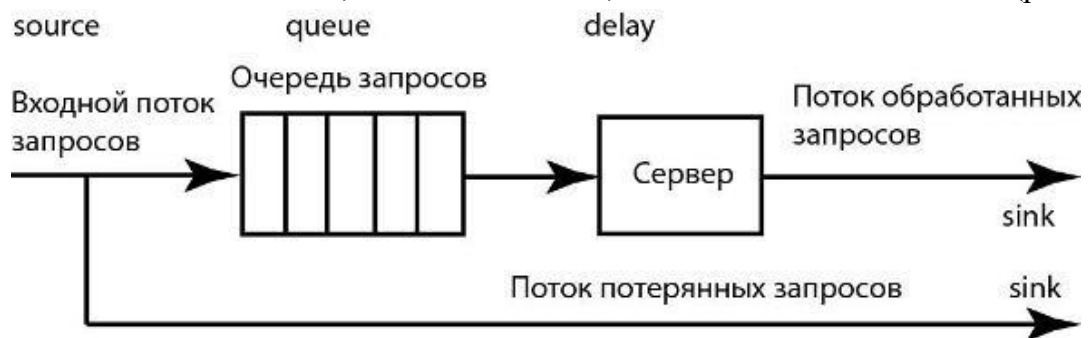


Рисунок 7.1 - Сервер как система массового обслуживания

В зависимости от исходных данных, выбранных согласно варианту, выданному преподавателем, построить имитационную модель для определения следующих показателей сервера:

- количество обработанных запросов;
- вероятность обработки запросов;
- среднее время обработки одного запроса;
- средняя длина очереди запросов к серверу;
- коэффициент использования сервера.

Индивидуальные исходные данные приведены в таблице 7.2

Количество прогонов модели определить по формуле :

$$N = t_{\alpha}^2 \cdot \frac{p \cdot (1 - p)}{\varepsilon^2},$$

где t_{α} - табулированный аргумент функции Лапласа (см. таблицу 7.3), p - ожидаемая вероятность исхода события, в данном случае вероятность обработки запросов сервером, ε - точность, вычислений.

ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ:

1. Разобраться с теоретическими основами решения задач СМО.
2. На основе примера выполнить индивидуальное задание согласно варианту, выданному преподавателем.
3. Определить как изменятся показатели работы сервера, если увеличить его производительность в 1,5-2 раза. Сделать вывод.

4. Определить как изменятся показатели работы сервера, если **увеличить или уменьшить** интенсивность поступления запросов в 2-3 раза. Сделать вывод. (**т.е. интервал между поступлениями запросов IntPostZ надо уменьшить или увеличить соответственно**)
5. Определить как изменятся показатели работы сервера, если увеличить емкость входного буфера в 2-3 раза. Сделать вывод.
6. Оформить отчет с описанием хода выполнения работы и копиями экранов, поясняющих действия. Сделать выводы по работе модели сервера.
7. Продемонстрировать работу преподавателю. Ответить на контрольные вопросы.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Система массового обслуживания (СМО) – математический (абстрактный) объект, содержащий один или несколько приборов П (каналов), обслуживающих заявки З, поступающие в систему, и накопитель Н, в котором находятся заявки, образующие очередь О и ожидающие обслуживания (рис.7.2).

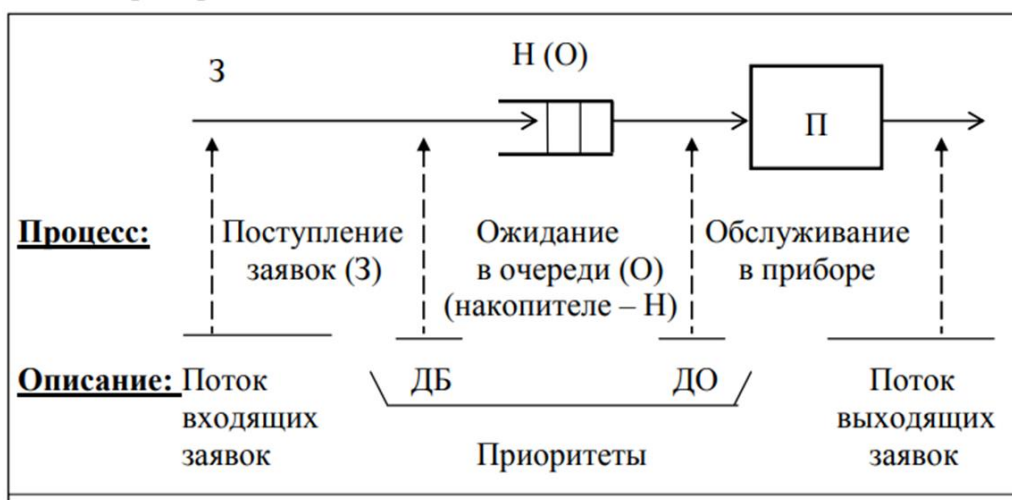


Рисунок 7.2 – Система массового обслуживания

Заявка (требование, запрос, вызов, клиент) – объект, поступающий в СМО и требующий обслуживания в обслуживающем приборе.

Совокупность заявок, распределенных во времени, образуют *поток заявок*.

Обслуживающий прибор или просто прибор (устройство, канал, линия) – элемент СМО, функцией которого является обслуживание заявок. В каждый момент времени в приборе на обслуживании может находиться только одна заявка.

Обслуживание – задержка заявки на некоторое время в обслуживающем приборе.

Длительность обслуживания – время задержки (обслуживания) заявки в приборе.

Накопитель (буфер) – совокупность мест для ожидания заявок перед обслуживающим прибором. Количество мест для ожидания определяет *ёмкость накопителя*.

Заявка, поступившая на вход СМО, может находиться в двух состояниях:

- в состоянии обслуживания (в приборе);
- в состоянии ожидания (в накопителе), если все приборы заняты обслуживанием других заявок.

Заявки, находящиеся в накопителе и ожидающие обслуживания, образуют *очередь заявок*. Количество заявок, ожидающих обслуживания в накопителе, определяет *длину очереди*.

Дисциплина обслуживания – правило выбора заявок из очереди для обслуживания в приборе.

Совокупность событий распределенных во времени называется *поток*. Если событие заключается в появлении заявок, имеем *поток заявок*.

Основной характеристикой потока заявок является его *интенсивность* λ – частота появления заявок или среднее число заявок, поступающих в СМО в единицу времени (математическое ожидание).

Величина $a = 1/\lambda$ определяет *средний интервал времени между двумя последовательными заявками*.

Длительность обслуживания $t_{обс}$ – время нахождения заявки в приборе – в общем случае величина случайная и описывается функцией или плотностью распределения. В случае неоднородной нагрузки длительности обслуживания заявок разных классов могут различаться законами распределений или только средними значениями. Часто длительность обслуживания заявок предполагается распределенной по экспоненциальному закону, что существенно упрощает аналитические выкладки.

Величина, обратная средней длительности обслуживания $t_{обс}$, характеризует среднее число заявок, которое может быть обслужено за единицу времени, и называется *интенсивностью обслуживания*: $\mu = 1/t_{обс}$. Во многих случаях аналитические зависимости могут быть получены для произвольного закона распределения длительности обслуживания заявок.

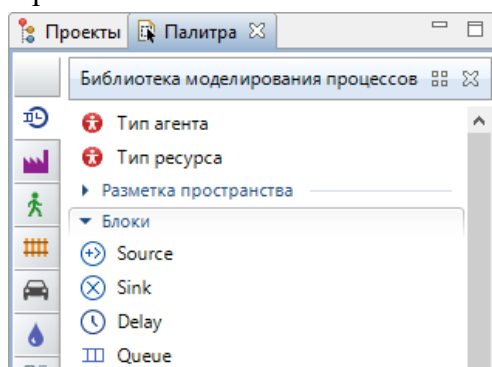
Среднеквадратическое отклонение (σ) показывает насколько велик разброс во время прихода событий относительно математического ожидания (λ). Именно дисперсия определяет случайность появления события, слабую предсказуемость моментов времени его появления.

Существует большое многообразие СМО, различающихся структурной и функциональной организацией. В то же время, разработка аналитических методов расчета характеристик функционирования СМО во многих случаях предполагает наличие ряда предположений, ограничивающих множество исследуемых СМО.

Большинство СМО, используемых в качестве базовых моделей реальных систем, могут быть классифицированы: по числу мест в накопителе (без накопителя – СМО с отказами; с накопителем ограниченной ёмкости – СМО с потерями; с накопителем неограниченной ёмкости – СМО без потерь); по количеству обслуживающих приборов (одноканальные и многоканальные); по количеству классов заявок (с однородным и неоднородным потоком заявок).


Основные возможности среды AnyLogic для построения и исследования СМО:

1. По умолчанию при создании новой модели в панели *Палитра* открывается *Библиотека моделирования процессов*. Вы можете открывать палитры щелчком по соответствующей иконке на вертикальной панели слева от палитры:



2. В основе каждой дискретно-событийной модели лежит диаграмма процесса – последовательность соединенных между собой объектов (*Библиотеки моделирования процессов*), задающих последовательность операций, которые будут производиться над проходящими по диаграмме процесса заявками.

Основные объекты (блоки) *Библиотеки моделирования процессов*, которые будут использоваться в модели работы сервера:

 Объект **Source** генерирует заявки определенного типа. Обычно он используется в качестве начальной точки диаграммы процесса, формализующей поток заявок. В модели работы

сервера заявками будут запросы на обработку сервером, а объект Source будет моделировать их поступление.

III Объект **Queue** моделирует очередь заявок, ожидающих приема объектами, следующими за данным в диаграмме процесса. В нашем случае он будет моделировать очередь запросов, ожидающих освобождения сервера.

Объект **Queue** моделирует очередь агентов, ожидающих приема блоками, следующими за данным в потоковой диаграмме, или же хранилище агентов общего назначения. При необходимости вы можете задать максимальное время ожидания агента в очереди. Вы также можете программно извлекать агентов из любых позиций в очереди.

Поступающие агенты помещаются в очередь в определенном порядке: либо согласно правилу FIFO (в порядке поступления в очередь - по умолчанию), LIFO, либо согласно приоритетам агентов. Приоритет может либо явно храниться в агенте, либо вычисляться согласно свойствам агента и каким-то внешним условиям. Очередь с приоритетами всегда примет нового входящего агента, вычислит его приоритет и поместит его в очередь в позицию, соответствующую его приоритету. Если очередь будет заполнена, то приход нового агента вынудит последнего хранящегося в очереди агента покинуть блок через порт **outPreempted** (но если приоритет нового агента не будет превышать приоритет последнего агента, то тогда вместо него будет вытеснена именно этот новый агент).

В режиме таймаута агент покинет очередь через порт **outTimeout**, если проведет в очереди заданное количество времени.

outPreempted outTimeout



Вы можете извлекать любых агентов из очереди с помощью функции **remove()**. В некоторых случаях, например, когда блок **Queue** используется для моделирования хранилища с произвольным доступом, имеет смысл оставлять порт **out** несоединенным и извлекать агентов из очереди с помощью функции **remove()**.

⌚ Объект **Delay** задерживает заявки на заданный период времени, представляя в нашей модели сервер, обрабатывающий запросы.

⊗ Объект **Sink** уничтожает поступившие заявки. Обычно он используется в качестве конечной точки потока заявок (и диаграммы процесса соответственно).

3. Чтобы добавить блоки *Библиотеки моделирования процессов* на диаграмму, перетащите требуемый элемент из палитры в графический редактор (окно Main).

Когда вы перетаскиваете блоки и располагаете их рядом друг с другом, вы можете видеть, как появляются соединительные линии между блоками. Будьте внимательны, эти линии должны соединять только порты, находящиеся с правой или левой стороны иконок.

4. Свойства объекта (как и любого другого элемента AnyLogic) можно изменить в панели **Свойства**.

Обратите внимание, что панель **Свойства** является контекстно-зависимой. Она отображает свойства выделенного в текущий момент элемента. Поэтому для изменения свойств элемента нужно будет предварительно щелчком мыши выделить его в графическом редакторе или в панели **Проекты**.


Чтобы всегда была уверенность в том, что в текущий момент в рабочем пространстве выбран именно нужный элемент, и именно его свойства вы редактируете в панели **Свойства**, обращайте внимание на первую строку, показываемую в панели **Свойства** - в ней отображается имя выбранного в текущий момент времени элемента и его тип.

Согласно принятым стандартам, объекты в диаграмме процесса обычно располагаются цепочкой слева направо, представляя собой последовательную очередность операций, которые будут производиться над заявкой.

5. *Настройка запуска модели и ее запуск.* Вы можете сконфигурировать выполнение модели в соответствии с вашими требованиями. Модель выполняется в соответствии с набором установок, задаваемым специальным элементом модели - экспериментом. Вы можете создать несколько экспериментов с различными установками и, изменять конфигурацию модели, просто запуская тот или иной эксперимент модели.

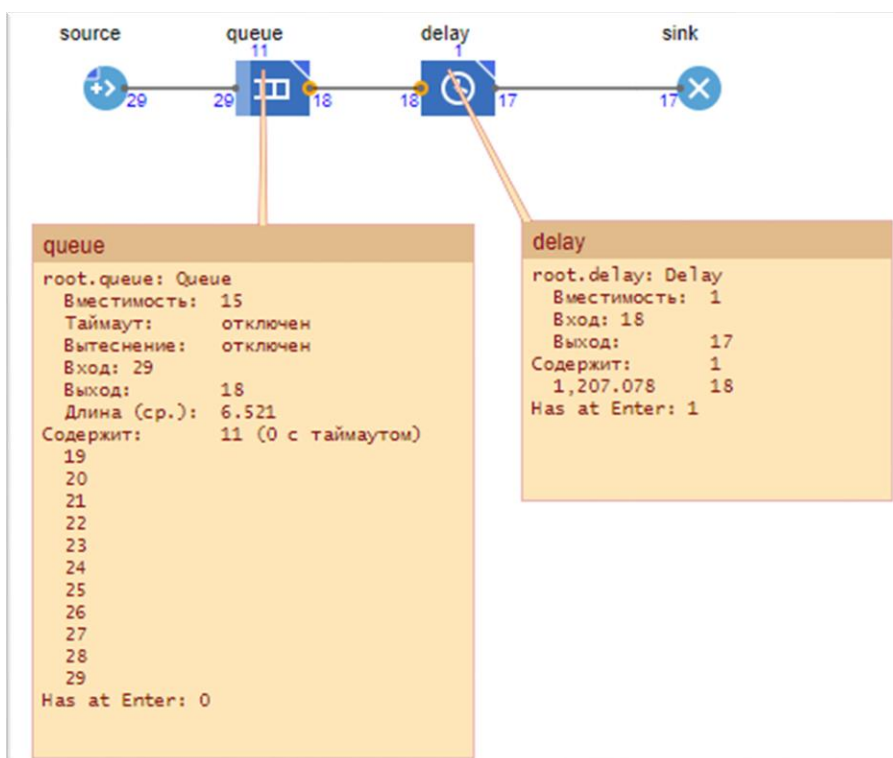
В панели *Проект* эксперименты отображаются в нижней части дерева модели. Один эксперимент, названный *Simulation*, создается по умолчанию. Это простой эксперимент, позволяющий запускать модель с заданными значениями параметров, поддерживающий режимы виртуального и реального времени, анимацию и отладку модели.

Если вы хотите наблюдать поведение модели в течение длительного периода (до того момента, пока вы сами не остановите выполнение модели), то по умолчанию времени остановки нет.


6. Построение созданной модели осуществляется с помощью кнопки панели инструментов  (F7). Чтобы построить модель в рабочей области AnyLogic должен быть выбран какой-то элемент именно этой модели.

Если в модели есть какие-нибудь ошибки, то построение не будет завершено, и в панель Ошибки будет выведена информация об ошибках, обнаруженных в модели. Двойным щелчком мыши по ошибке в этом списке вы можете перейти к предполагаемому месту ошибки, чтобы исправить её. При этом откроется соответствующее место ошибки.

7. Следить за состоянием любого объекта диаграммы процесса во время выполнения модели возможно с помощью окна инспекта этого объекта. Чтобы открыть окно инспекта, щёлкните мышью по значку нужного блока. Окно инспекта, подведя курсор, можно перемещать в нужное вам место. Также, подведя курсор к правому нижнему углу окна инспекта, можно при необходимости изменять его размеры:



В окне инспекта будет отображена базовая информация по выделенному объекту: например, для объекта *queue* будут отображены вместимость очереди, количество заявок, прошедшее через каждый порт объекта и т. д. Такая же информация содержится в инспекте и для объекта *delay*.

8. Когда вы захотите остановить выполнение модели, Щёлкните мышью кнопку **Прекратить выполнение**  панели управления окна презентации.

ПРИМЕР ВЫПОЛНЕНИЯ РАБОТЫ В СРЕДЕ ANYLOGIC

Постановка задачи

Сервер обрабатывает запросы, поступающие с автоматизированных рабочих мест с интервалами, распределенными по показательному закону со средним значением $TI = 2$ мин. Сервер имеет входной буфер ёмкостью 5 запросов. В случае полной занятости входного буфера поступающий запрос теряется.

Вычислительная сложность запросов подчинена нормальному закону с математическим ожиданием $SI = 6 \cdot 10^7$ оп и среднеквадратическим отклонением $S2 = 2 \cdot 10^5$ оп. Производительность сервера $Q = 6 \cdot 10^5$ оп/с.

Построить имитационную модель обработки запросов сервером для определения оценки математического ожидания количества запросов (далее - количества запросов), обработанных сервером за время функционирования $T = 1$ час, и оценки математического ожидания вероятности обработки запросов (далее - вероятности обработки запросов).

Привести ряд экспериментов с помощью построенной модели и определить:

- как изменятся показатели работы сервера, если увеличить его производительность в 1,5-2 раза. Сделать вывод.
- как изменятся показатели работы сервера, если увеличить интенсивность поступления запросов в 2-3 раза. Сделать вывод.
- как изменятся показатели работы сервера, если увеличить ёмкость входного буфера в 2-3 раза. Сделать вывод.

Уяснение задачи моделирования

Сервер представляет собой однофазную систему массового обслуживания разомкнутого типа с ограниченной входной ёмкостью, то есть с отказами, и абсолютной надёжностью (рис.7.1).

В модели должны быть следующие элементы:

- задание исходных данных;
- описание арифметических выражений;
- сегмент имитации поступления и обработки запросов;
- сегмент задания времени моделирования и расчета результатов моделирования.

Серверу дадим имя Server. Для счета количества всех запросов используем метку KolZap.

Выберем масштаб: 1 единице масштабного времени соответствует 1 с. Так как среднее значение интервалов поступления запросов, $TI = 2$ мин. то теперь это будет 120 ед. мод. времени.

Для начала построим модель, где примем, что время обработки сервером одного запроса распределено по экспоненциальному закону со средним значением 3 мин. (180 ед. мод. времени), а затем добавим параметры сервера, указанные в условии задачи.

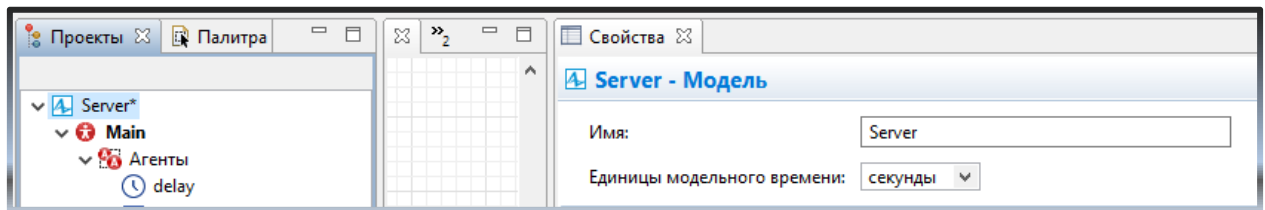
Рассчитаем количество прогонов, которые нужно выполнить в каждом наблюдении, т.е. проведем так называемое тактическое планирование эксперимента. Пусть результаты моделирования (вероятность обработки запросов) нужно получить с доверительной вероятностью $\alpha = 0,95$ и точностью $\varepsilon = 0,01$. Расчет проведем для худшего случая, т.е. при вероятности $p = 0,5$, так как до эксперимента p неизвестно:

$$N = t_{\alpha}^2 \cdot \frac{p \cdot (1 - p)}{\varepsilon^2} = 1,96^2 \cdot \frac{0,5 \cdot (1 - 0,5)}{0,01^2} = 3,8416 \cdot \frac{0,25}{0,0001} = 9604$$

где $t_{\alpha} = 1,96$ - табулированный аргумент функции Лапласа (см. таблицу 7.3).

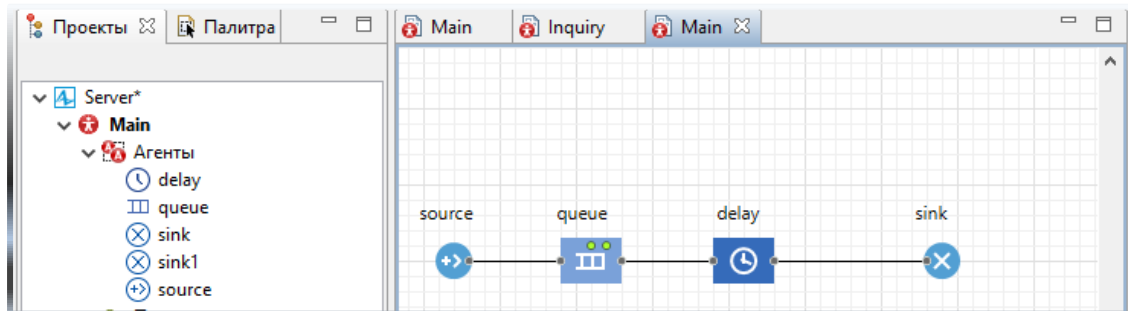
Создание новой модели

- Выполните команду **Файл/Создать/Модель** на **Панели инструментов**. Откроется диалоговое окно **Новая модель**.
- В поле **Имя модели** диалогового окна **Новая модель** введите **Server**. Выберите каталог, в котором будут сохранены файлы модели.
- Укажите **единицы модельного времени** – секунды.



Создание диаграммы процесса

-Для создания диаграммы процесса в *Палитре* выделите *Библиотеку моделирования процессов*. Из неё перетащите объекты на диаграмму и соедините, как показано на рисунке ниже:



Для добавления объекта на диаграмму, надо щёлкнуть его мышкой и, не отпуская её, перетащить в графический редактор. При перетаскивании объектов вы можете видеть, как появляются соединительные линии между объектами.



Обращайте внимание на то, чтобы эти линии, если необходимо, соединяли только порты, находящиеся с правой или левой стороны иконок. Если у вас не получится автоматическое соединение нужных объектов, дважды щёлкните начальный порт. Он станет зелёным. Протащите курсор к конечному порту. Он также станет зелёным. Для завершения процесса соединения дважды щёлкните конечный порт.

Изменение свойств блоков модели, её настройка и запуск

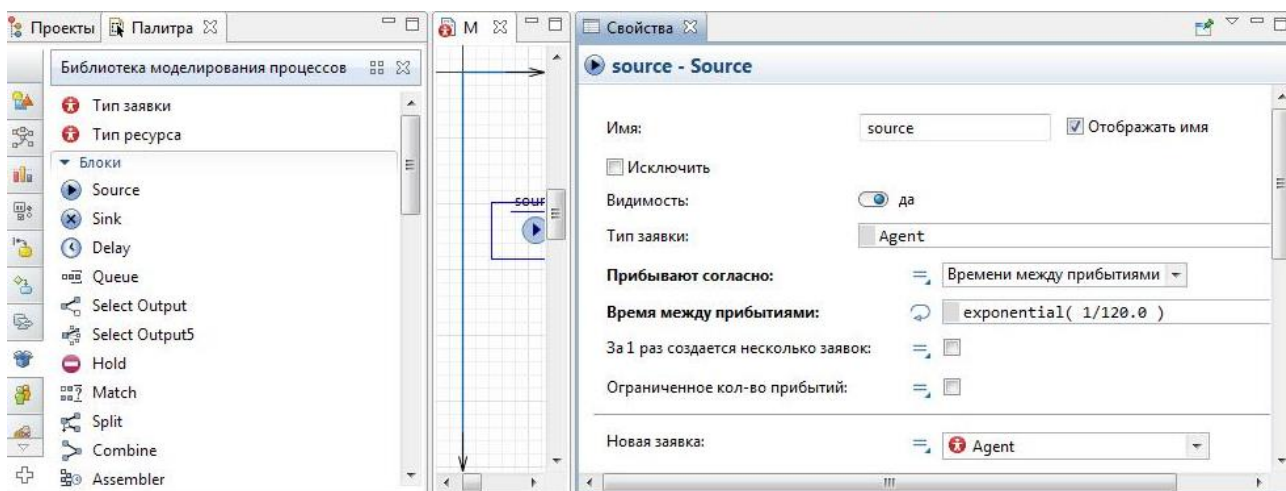
Всё, что нам нужно, чтобы сделать созданную диаграмму модели адекватной постановке задачи - это изменить некоторые свойства объектов.


Изменение свойств блоков диаграммы процесса:


1) Первым объектом в диаграмме процесса является объект класса **Source**.

По заданию объект создает заявки через временной интервал, распределенный по показательному (экспоненциальному) закону со средним значением 2 мин.

- Установите среднее время поступления запросов и среднее время их обработки в секундах.
- Выделите объект **source**. В выпадающем списке **Прибывают согласно**: укажите, что запросы поступают согласно **Времени между прибытиями**.
- В поле **Время между прибытиями** появится запись **exponential(1)**. Установите согласно постановке задачи среднее значение интервалов времени поступления запросов на сервер, изменив свойства объекта **source**. Для этого вместо характеристики распределения 1 введите **1/120.0**.



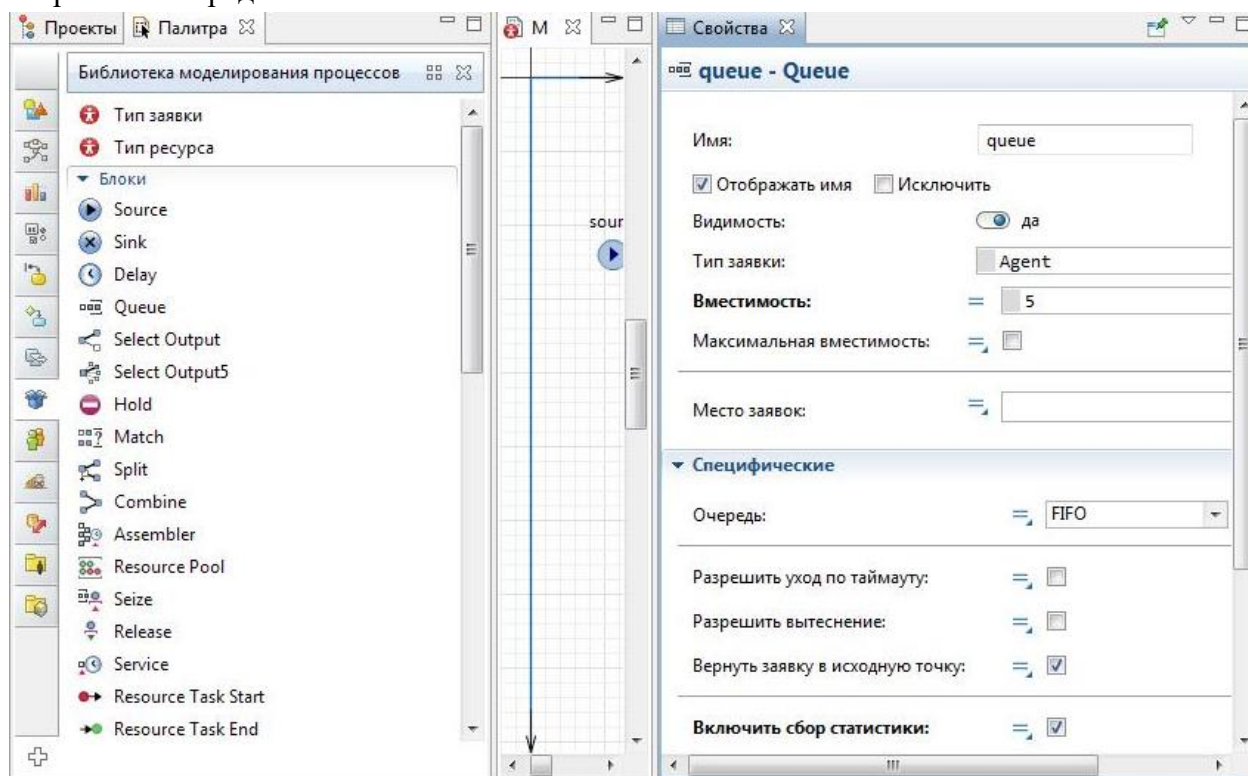
 Запись 1/120.0 необходима потому, что аргументом функции `exponential` является интенсивность поступления заявок, а не средний интервал времени поступления запросов. В выпадающем списке выберите секунды.


 В языке программирования Java символ `/` означает целочисленное деление, т.е. если оба числа целые, то и результат будет целым. В нашем случае отношение 1/120 было бы равно нулю. Для получения вещественного результата, необходимо, чтобы хотя бы одно из чисел было вещественным (`double`). Поэтому в качестве характеристики экспоненциального распределения (интенсивности поступления запросов) необходимо указать 1/120.0 или 1.0/120.

2) Следующий объект - `queue`. Выделите его. Измените свойства объекта `queue`:

- Задайте длину очереди. Введите в поле **Вместимость**: 5. В очереди будут находиться не более 5 запросов.

- Установите флажок **Включить сбор статистики**, чтобы включить сбор статистики для этого объекта. В этом случае по ходу моделирования будет собираться статистика по количеству запросов в очереди.



 Если же вы не установите этот флажок, то данная функциональность будет недоступна, поскольку по умолчанию она отключена для повышения скорости выполнения модели. Для вывода, например, средней длины очереди, нужно в модели предусмотреть Java код.

3) Следующим в нашей диаграмме процесса расположен объект **delay**. Измените свойства объекта **delay** :

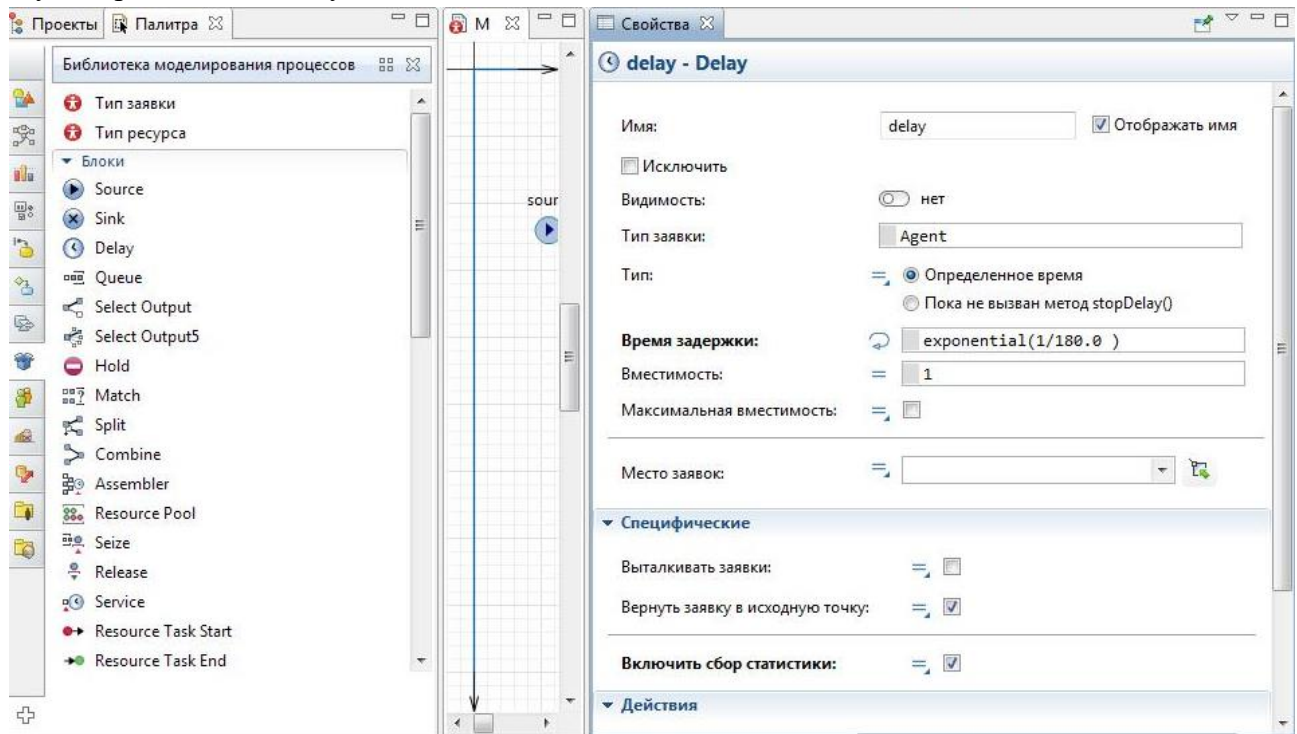
- Обработка одного запроса занимает примерно 3 мин. Задайте время обслуживания, распределенное по экспоненциальному закону со средним значением **3 мин**. Для этого введите в поле Время задержки: **exponential(1/180.0)**.



Функция **exponential()** является стандартной функцией генератора случайных чисел AnyLogic. AnyLogic предоставляет функции и других случайных распределений, таких как нормальное, треугольное, и т. д.

- Установите флажок **Включить сбор статистики**.

Для вывода коэффициента использования объекта **delay** в модели также следует предусмотреть соответствующий Java код.



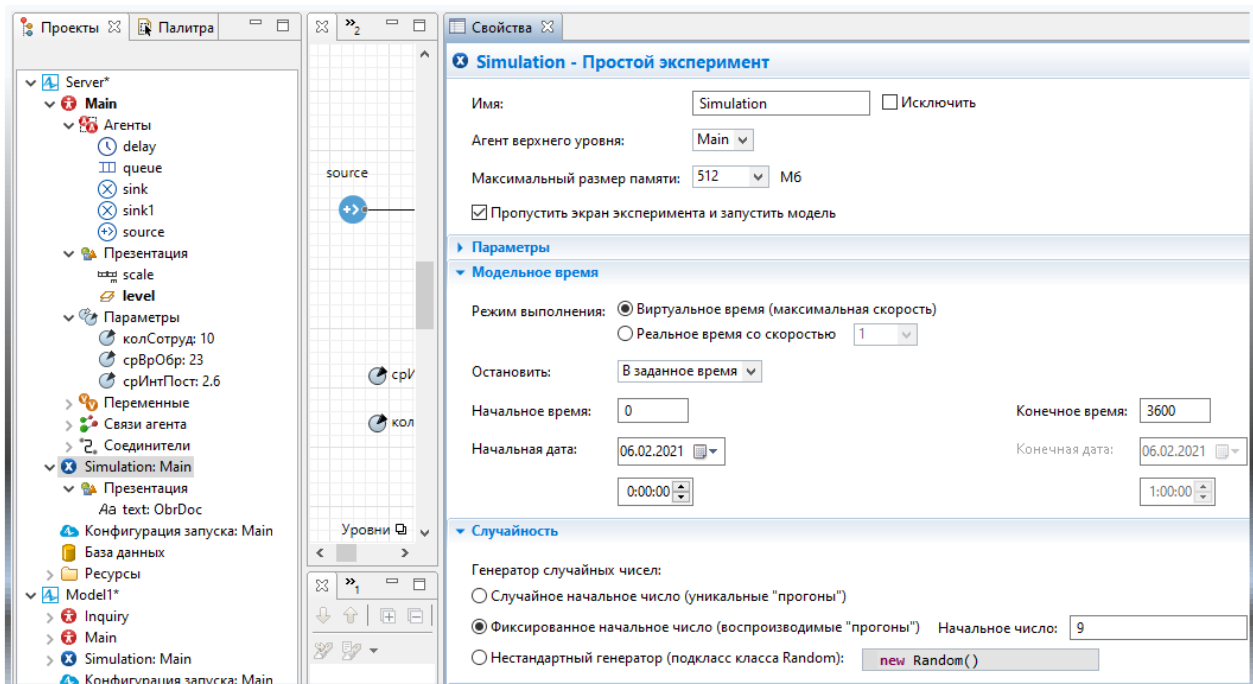
4) Последним в диаграмме нашей дискретно-событийной модели находится объект **sink**. В рассматриваемой модели его настройка не требуется.

Настройка запуска модели

Обработку запросов сервером мы планируем исследовать в течение одного часа, т.е. **3600 с**.

В панели **Проект** выделите эксперимент **Simulation:Main**:

- Щелчком раскройте вкладку **Модельное время**.
- Установите **Виртуальное время** (максимальная скорость).
- В поле **Остановить**: выберите из списка **В заданное время**.
- В поле **Конечное время**: установите **3600**.
- Раскройте вкладку **Случайность**.
- Выберите опцию **Фиксированное начальное число** (воспроизводимые прогоны).
- В поле **Начальное число**: установите **9**.



Запуск модели

- Постройте вашу модель с помощью кнопки **панели инструментов** (F7).

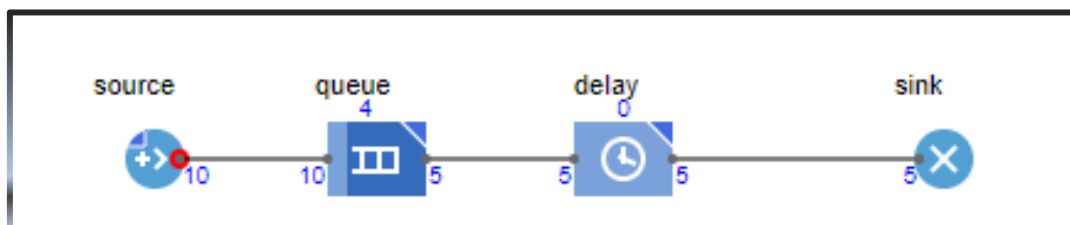
- Щёлкните мышью кнопку панели инструментов **Запустить** (или нажмите F5) и выберите из открывшегося списка эксперимент, который вы хотите запустить. Эксперимент этой модели будет называться **Server/Simulation**. После запуска модели вы увидите окно презентации этой модели. В нем будет отображена презентация запущенного эксперимента-



В дальнейшем нажатием кнопки **Запустить** (или кнопки F5) будет запускаться тот эксперимент, который запускался вами в последний раз. Чтобы выбрать другой эксперимент, вам будет нужно щёлкнуть мышью по стрелке, находящейся в правой части кнопки **Запустить**, и выбрать нужный вам эксперимент из открывшегося списка (или щёлкнуть правой кнопкой мыши по этому эксперименту в панели **Проект** и выбрать **Запустить** из контекстного меню).

- Для каждого объекта определены правила, при каких условиях принимать заявки. Некоторые объекты задерживают заявки внутри себя, некоторые - нет. Для объектов также определены правила: может ли заявка, которая должна покинуть объект, ожидать на выходе, если следующий объект не готов её принять.

Если заявка должна покинуть объект, а следующий объект не готов её принять, и заявка не может ждать, то модель останавливается с ошибкой.



Ошибка означает, что запрос не может покинуть объект **source** и войти в блок **queue**, так как его ёмкость, равная **5**, заполнена. Также выдаётся сообщение о логической ошибке в модели:

```
Ошибка при выполнении дискретного события:
Логическая ошибка в модели:
root.source:
Агент не смог покинуть этот порт: root.source.out в момент времени 642.571 / дату
com.anylogic.engine.ModelException: root.source:
Агент не смог покинуть этот порт: root.source.out в момент времени 642.571 / дату
```

- Измените свойства объекта **queue**, т. е. увеличьте длину очереди. Для этого введите в поле Вместимость **15**. Снова запустите модель.

(Можете убедиться, что при увеличении ёмкости в пределах 6 ... 14 модель по-прежнему останавливается с этой же ошибкой.) Момент появления ошибки зависит от длительности времени моделирования.



Для предотвращения остановок модели по ранее указанной ошибке - недостаточной ёмкости объекта **queue** - мы увеличили ёмкость объекта **queue**. Однако можно было бы изменять среднее время имитации поступления запросов объектом **source** и среднее время обработки запросов сервером, т. е. среднее время задержки объекта **delay**, оставляя неизменной длину очереди и добиваясь безошибочной работы модели.

Конечно, при изменении свойств объектов модели нужно обязательно исходить из целей её построения. Мы же не выполнили условий, указанных в постановке задачи, поэтому к выполнению их вернемся позже.

Создание анимации модели

Можно было наблюдать, анализировать и интерпретировать работу запущенной модели с помощью визуализированной диаграммы процесса.

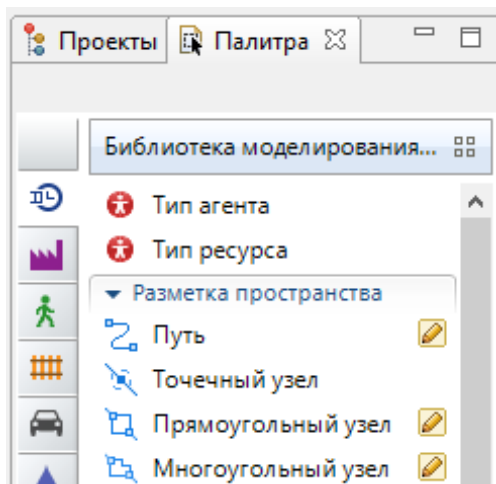
Однако удобнее в ряде случаев иметь более наглядную визуализацию с помощью анимации. В этой задаче мы хотим создать визуализированный процесс поступления запросов на сервер и обработки запросов сервером.

Так как в данном случае нас не интересует конкретное расположение объектов в пространстве, то мы можем просто добавить схематическую анимацию интересующих нас объектов - сервер и очередь запросов к нему.

Анимация модели рисуется в той же диаграмме (в графическом редакторе), в которой задается и диаграмма моделируемого процесса:

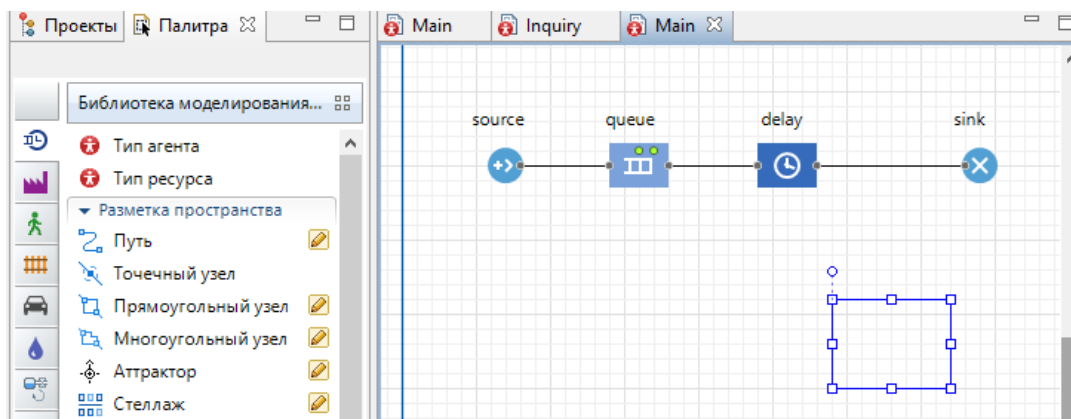
1. Нарисуйте прямоугольный узел, который будет обозначать на анимации сервер.

- Откройте палитру **Разметка пространства**. Чтобы открыть какую-либо палитру, нужно щелкнуть по иконке этой палитры.



Палитра **Разметка пространства** содержит в качестве элементов различные примитивные фигуры, используемые для рисования презентаций моделей. Это путь, прямоугольный узел, многоугольный узел, точечный узел, аттрактор, стеллаж, масштаб.

- Выделите элемент **Прямоугольный узел** и перетащите его на диаграмму класса активного объекта. Поместите элемент **Прямоугольный узел** так, как показано на рисунке ниже:



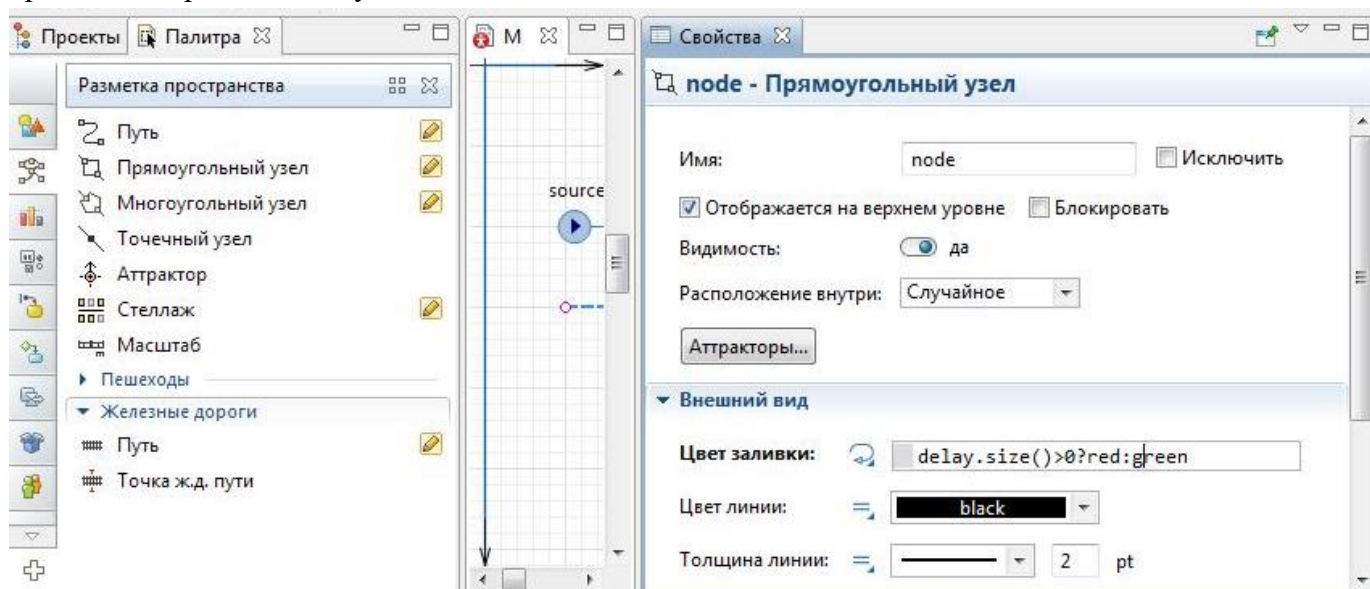
2. Сделаем так, чтобы цвет этого прямоугольного узла будет меняться в зависимости от того, обрабатывает ли сервер в данный момент времени запрос или нет.

- Для этого выделите нарисованную нами фигуру на диаграмме. Перейдите на страницу [Внешний вид](#) панели свойств.

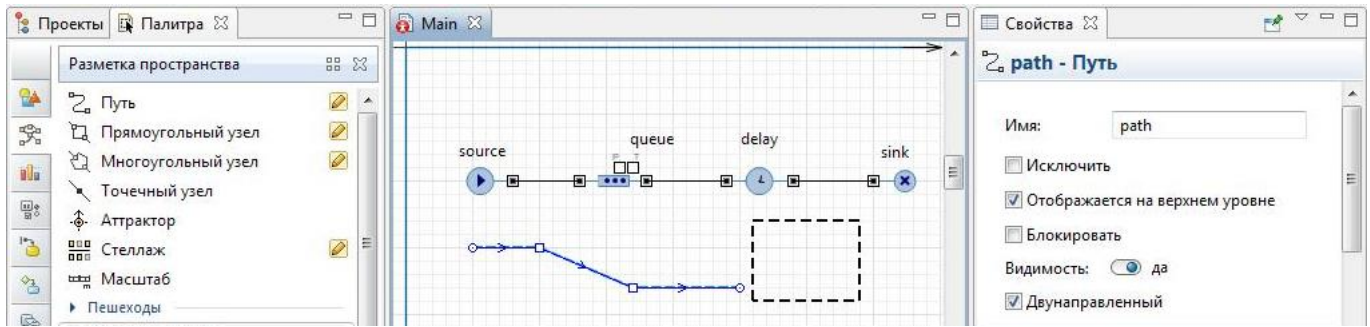
Если нужно, чтобы по ходу моделирования то или иное свойство фигуры меняло своё значение в зависимости от каких-то условий, то можете ввести в поле соответствующего динамического свойства выражение, которое будет постоянно вычисляться заново при выполнении модели. Возвращаемый результат вычисления будет присваиваться текущему значению этого свойства.

- Мы хотим, чтобы во время моделирования менялся цвет нашей фигуры, поэтому щёлкните в поле **Цвет заливки**: по стрелке, выберите **Динамическое значение** и введите там следующую строку: `delay.size()>0?red:green`


Здесь `delay` - это имя нашего объекта `delay`. Функция `size()` возвращает число запросов, обслуживаемых в данный момент времени. Если сервер занят, то цвет прямоугольника будет красным, в противном случае - зелёным.



3. Нарисуйте путь, который будет обозначать на анимации очередь к серверу.

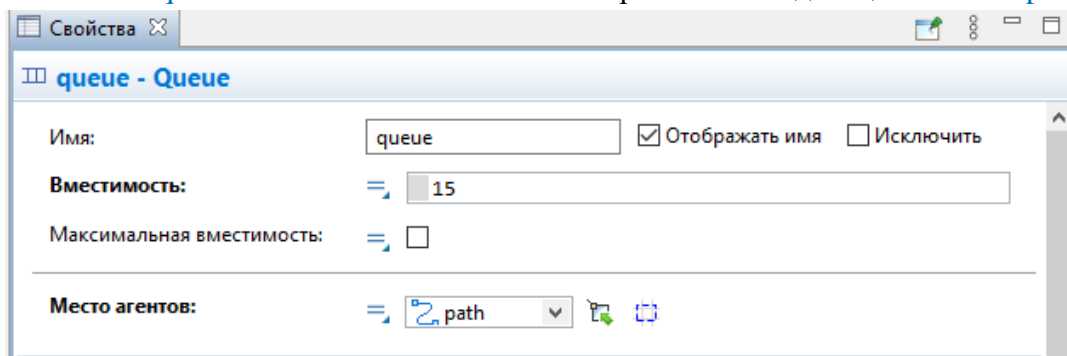


Чтобы нарисовать путь, сделайте двойной щелчок мышью по элементу **Путь** палитры **Разметка пространства**, чтобы перейти в режим рисования. Теперь вы можете рисовать путь точка за точкой, последовательно щелкая мышью в тех точках диаграммы, куда вы хотите поместить вершины пути. Чтобы завершить рисование, добавьте последнюю точку пути двойным щелчком мыши.

 Очень важно, какую точку пути вы создаете первой. Заявки будут располагаться вдоль нарисованного вами пути в направлении от конечной точки к начальной точке. Поэтому обязательно начните рисование пути слева и поместите рядом с сервером конечную точку пути, которая будет соответствовать в этом случае началу очереди.

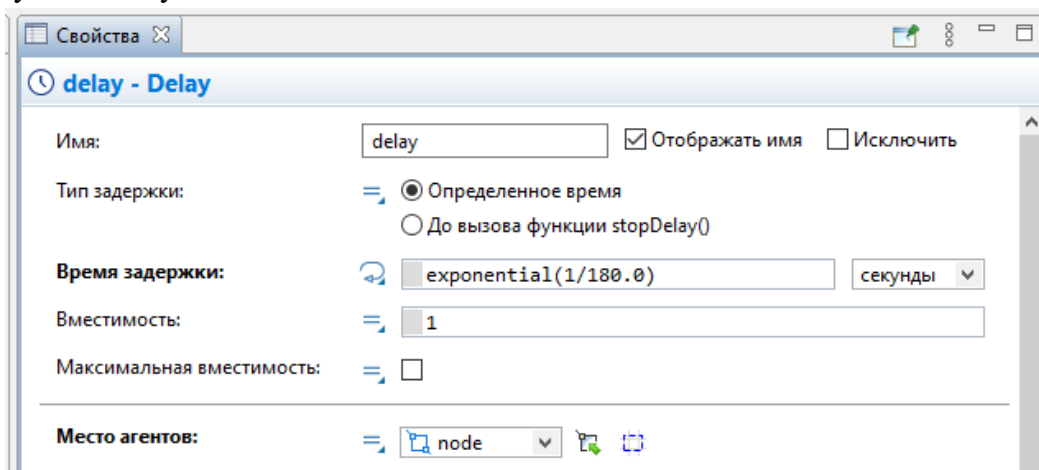
4. Теперь мы должны задать созданные анимационные объекты в качестве анимационных фигур объектов диаграммы нашего процесса.

Задайте путь в качестве фигуры анимации очереди. Выделите объект **queue**. На странице свойств объекта **queue** в поле **Место агентов**: выберите из выпадающего списка **path**:



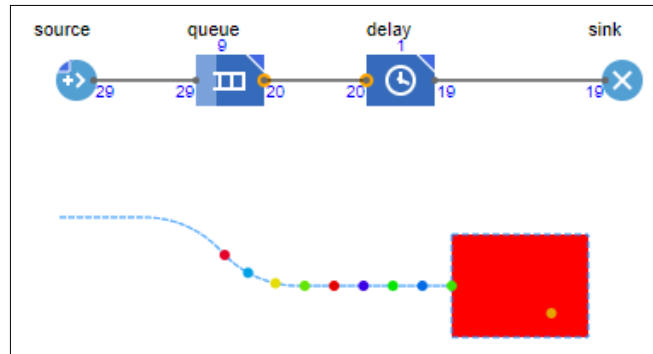
5. Задайте прямоугольный узел в качестве фигуры анимации сервера.

Выделите объект **delay**. Введите в поле **Место агентов**: из выпадающего списка имя нашего прямоугольного узла: **node**:



6. Запустите модель.

Вы увидите, что у модели теперь есть простейшая анимация - сервер и очередь запросов к нему. Цвет фигуры сервера будет меняться в зависимости от того, обрабатывается ли запрос в данный момент времени или нет.



Сбор статистики использования ресурсов

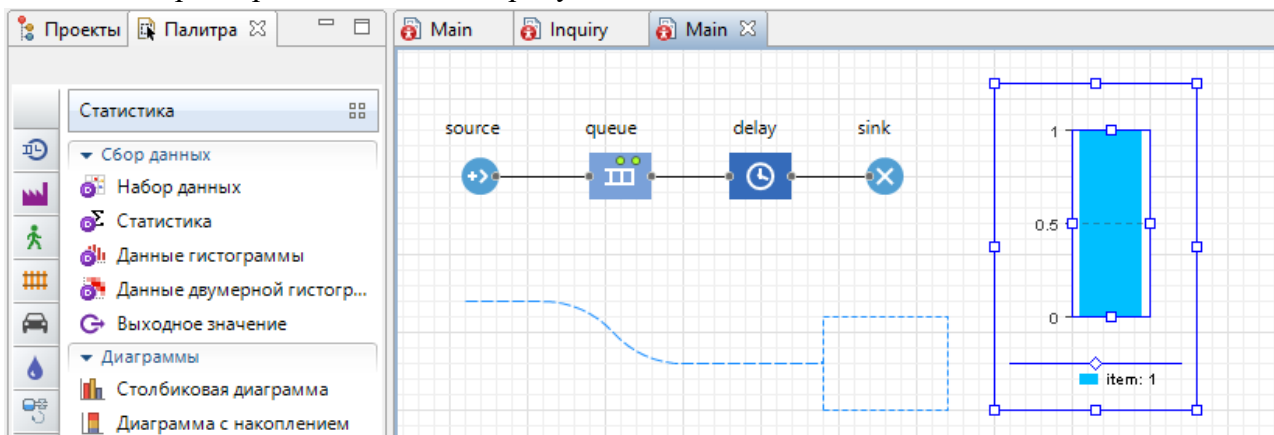
AnyLogic предоставляет пользователю удобные средства для сбора статистики по работе блоков диаграммы процесса. Объекты [Enterprise Library](#) самостоятельно производят сбор основной статистики. Все, что вам нужно сделать - это включить сбор статистики для объекта.

Поскольку мы уже сделали это для объектов [delay](#) и [queue](#), то теперь мы можем, например, посмотреть интересующую нас статистику (скажем, статистику занятости сервера и длины очереди) с помощью диаграмм.

1. Добавьте диаграмму для отображения среднего коэффициента использования сервера:

- Откройте палитру [Статистика](#). Эта палитра содержит элементы сбора данных и статистики, а также диаграммы для визуализации данных и результатов моделирования.

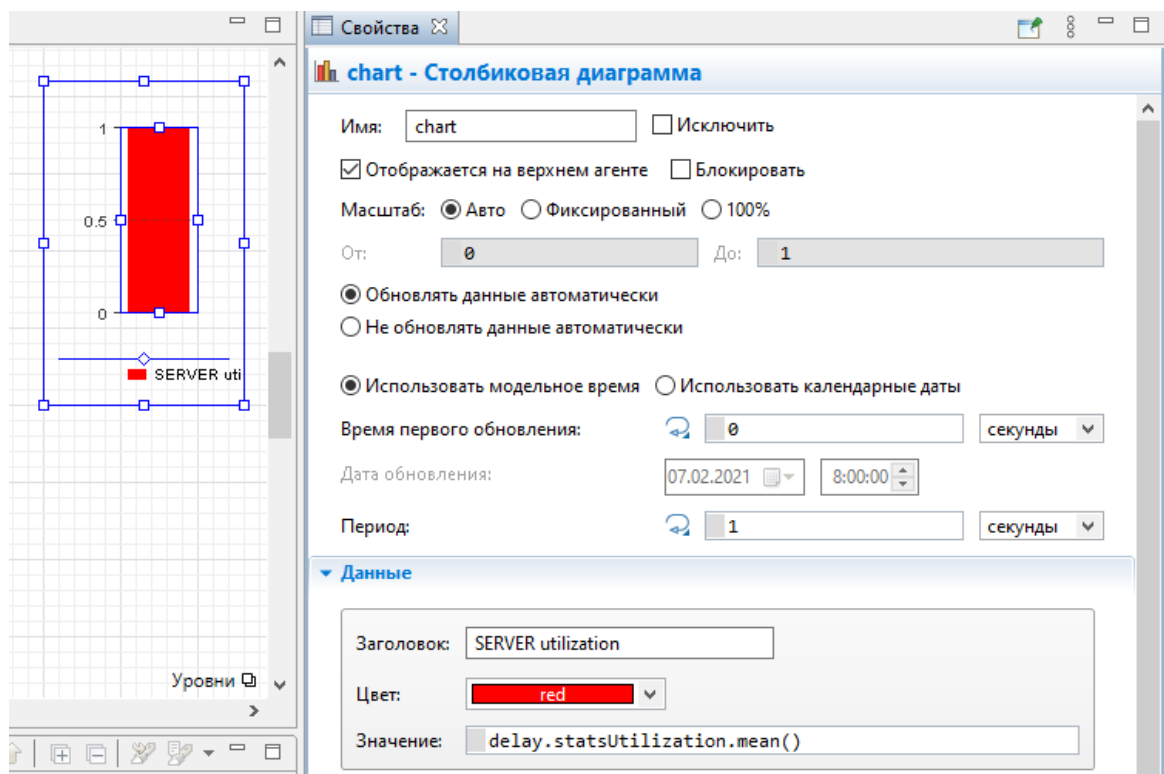
- Перетащите элемент [Столбиковая диаграмма](#) из палитры [Статистика](#) на диаграмму класса и измените ее размер, как показано на рисунке ниже:



- Перейдите на панель [Свойства](#). Щёлкните кнопку [Добавить элемент данных](#). После щелчка появится секция свойств того элемента данных ([chart - Столбиковая диаграмма](#)), который будет отображаться на этой диаграмме.

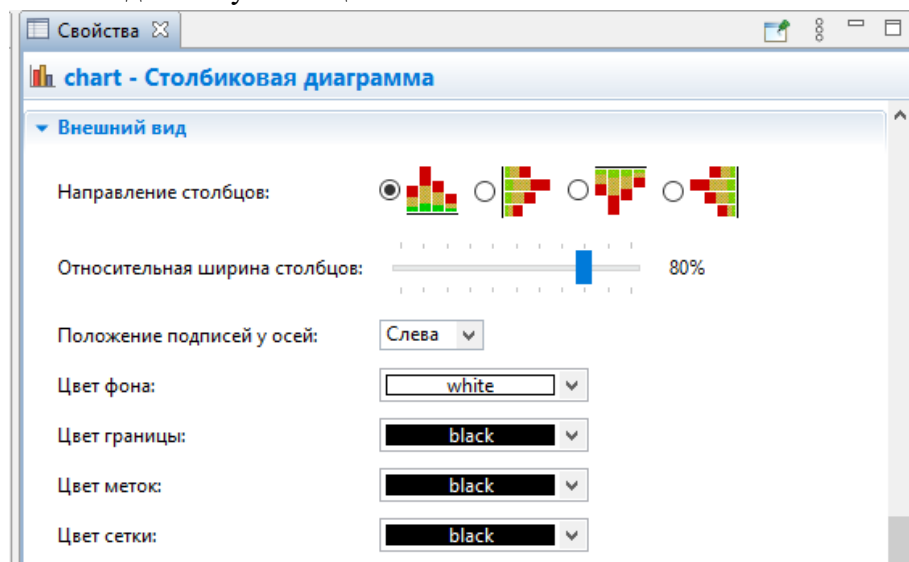
- Измените [Заголовок](#) на [SERVER utilization](#).

- Введите `delay.statsUtilization.mean()` в поле [Значение](#). Здесь [delay](#) - это имя нашего объекта [delay](#). У каждого объекта [delay](#) есть встроенный набор данных [statsUtilization](#), занимающийся сбором статистики использования этого объекта.



Функция `mean()` возвращает среднее из всех измеренных этим набором данных значений. Вы можете использовать и другие методы сбора статистики, такие, как `min()` или `max()`. Полный список методов можно найти на странице документации этого класса набора данных: `StatisticsContinuous` (на английском языке).

- Щёлкните **Внешний вид**. Установите свойства: направление столбцов, цвета фона, границ, меток, сетки, положение подписей у столбцов.



- Раскройте щелчками страницы (вкладки) **Местоположение и размер**, **Легенда**, **Область диаграммы**. Установите свойства, чтобы изменить расположение легенды относительно диаграммы (мы хотим, чтобы она отображалась внизу), размер диаграммы, высоту, ширину, координаты размещения на диаграмме, цвета текста, границы.

chart - Столбиковая диаграмма

Местоположение и размер

Уровень:

X: Ширина:

Y: Высота:

Легенда

☒ Отображать легенду

Высота:

Цвет текста:

Расположение: ☒ ☐ ☐ ☐

Область диаграммы

Смещение по оси X: Ширина:

Смещение по оси Y: Высота:

Цвет фона:

Цвет границы:

2. Аналогичным образом добавьте еще одну столбиковую диаграмму для отображения средней длины очереди.

- На панели **Свойства** щёлкните **Добавить элемент данных**. После щелчка появится страница **Данные** свойств элемента данных (**chart1 - Столбиковая диаграмма**), который также будет отображаться на этой диаграмме.

- В поле **Заголовок**: введите **Queue length**, а в поле **Значение**: введите **queue.statsSize.mean()**.

The screenshot shows a software interface with a process flow diagram on the left and a properties panel on the right. The process flow diagram includes a queue element labeled 'Queue length: 1'. The properties panel for 'chart1 - Столбиковая диаграмма' is open, showing the 'Данные' (Data) tab. The 'Заголовок' (Title) is 'Queue length' and the 'Значение' (Value) is 'queue.statsSize.mean()'. The 'Цвет' (Color) is set to 'peru'.

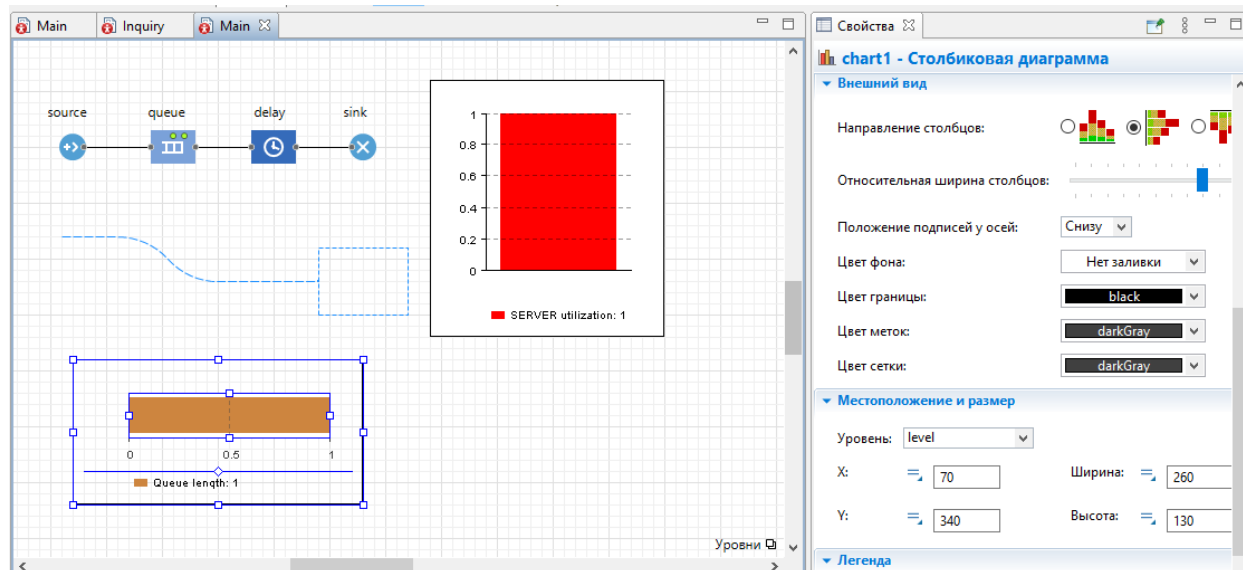


В поле **Значение**: `queue` - это имя нашего объекта `queue`. У каждого объекта `queue`, как и объекта `delay`, также есть встроенный набор данных `statsSize`, занимающийся сбором статистики

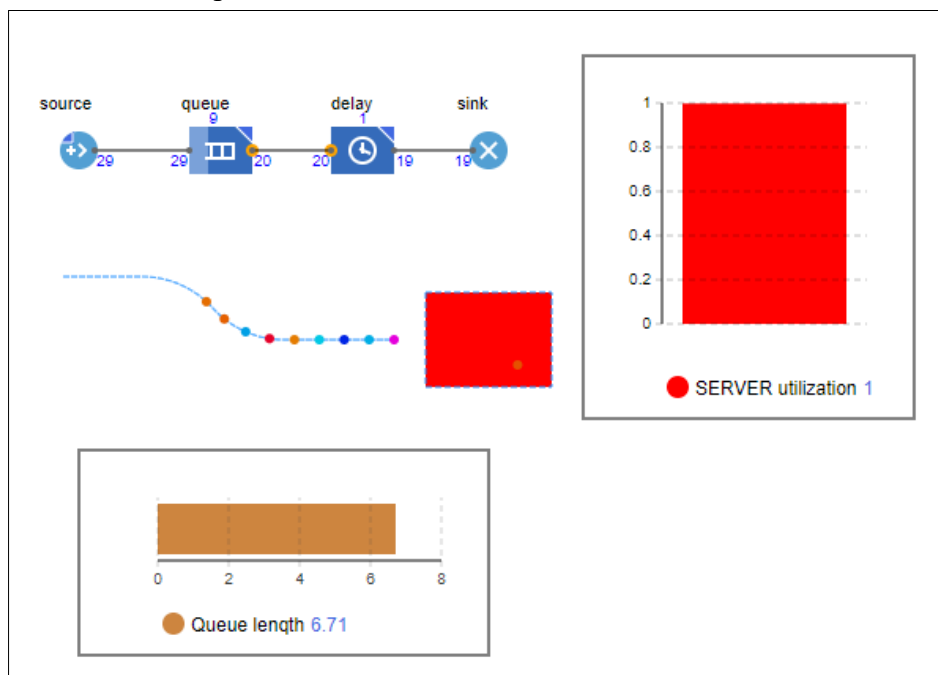
использования этого объекта. Функция `mean()` также возвращает среднее из всех измеренных этим набором данных значений.

- На страницах **Внешний вид**, **Местоположение и размер**, **Легенда**, **Область диаграммы** установите свойства самостоятельно.

- На странице **Внешний вид** панели **Свойства** выберите в секции свойств **Направление столбцов** вторую опцию, чтобы столбцы во второй столбиковой диаграмме, расположенной горизонтально, росли вправо:



- Запустите модель с двумя столбиковыми диаграммами, установив модельное время 3600 единиц, и наблюдайте за её работой.



Уточнение модели согласно ёмкости входного буфера

На работающей модели видно, что длина очереди равна 9-14 запросам при установленной максимальной длине 15. Но ведь в постановке задачи ёмкость буфера была определена в 5 запросов. Нам не удалось до этого построить модель с такой ёмкостью из-за ошибки - невозможности очередного запроса покинуть блок `source`, так как длина очереди уже была равна 5 запросам. Нам пришлось во избежание этой ошибки увеличить ёмкость буфера до 15 запросов.

А возможно ли выполнить данное условие постановки задачи средствами AnyLogic? Оказывается, что можно. Причем, различными способами.

Уточним модель согласно постановке задачи одним из этих способов.



Объект **queue** моделирует очередь заявок, ожидающих приёма объектами, следующими за ним в потоковой диаграмме, или же моделирует хранилище заявок общего назначения. При необходимости вы можете задать максимальное время ожидания заявки в очереди. Вы также можете с помощью написанной вами программы извлекать заявки из любых позиций в очереди.



Заявка может покинуть объект **queue** различными способами:

- обычным способом через порт **out**, когда объект, следующий в блок-схеме за этим объектом, готов принять заявку;
- через порт **outTimeout**, если заявка проведет в очереди заданное количество времени (если включен режим таймаута);
- через порт **outPreempted**, будучи вытесненной другой поступившей заявкой при заполненной очереди (если включен режим вытеснения);
- "вручную", путем вызова функции `remove()` или `removeFirst()`.

В первом случае объект **queue** покидает заявка, находящаяся в самом начале очереди (в нулевой позиции). Если заявка направлена в порт **outTimeout** или **outPreempted**, то она должна покинуть объект мгновенно. Если включена опция вытеснения, то объект **queue** всегда готов принять новую заявку, в противном случае при заполненной очереди заявка принята не будет.

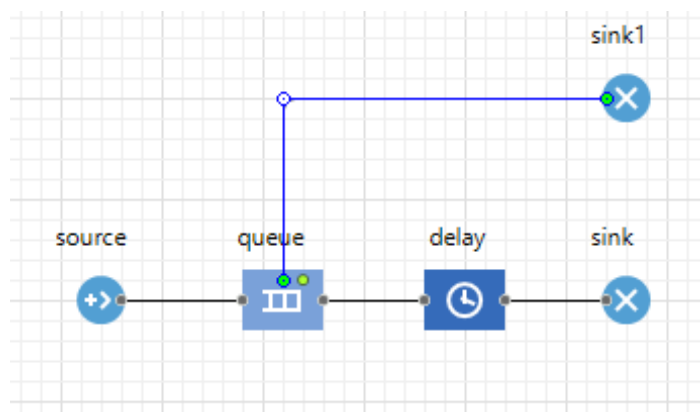


Поступающие заявки помещаются в очередь в определенном порядке: либо согласно правилу FIFO (в порядке поступления в очередь), либо согласно приоритетам заявок. Приоритет может быть либо явно храниться в заявке, либо вычисляться согласно свойствам заявки и каким-то внешним условиям. Очередь с приоритетами всегда примет новую входящую заявку, вычислит её приоритет, и поместит в очередь в позицию, соответствующую её приоритету. Если очередь будет заполнена, то приход новой заявки вынудит последнюю хранящуюся в очереди заявку покинуть объект через порт **outPreempted**. Но если приоритет новой заявки не будет превышать приоритет последней заявки, то тогда вместо неё будет вытеснена именно эта новая заявка.

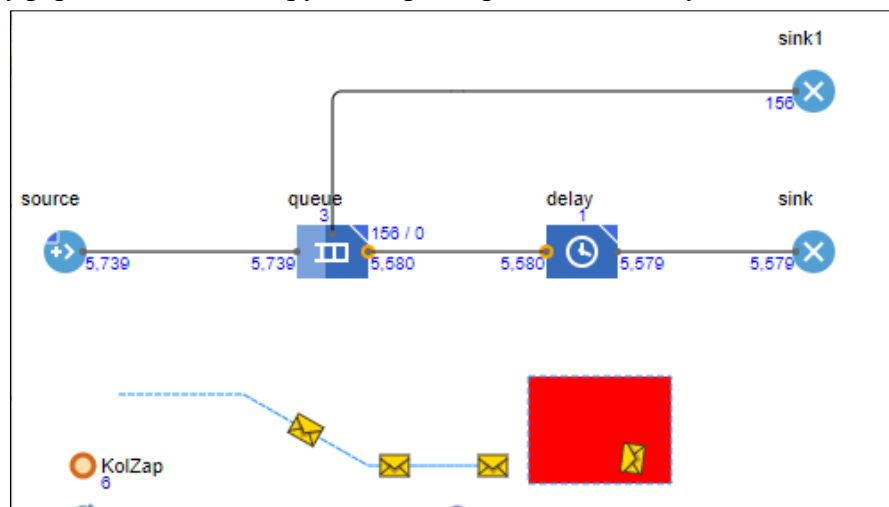
Для выполнения условия постановки задачи воспользуемся способом вытеснения. Все запросы, вырабатываемые объектом **source**, имеют один и тот же приоритет. Поэтому при полном заполнении накопителя (5 запросов) теряться будет последний запрос.

1. Уточните модель:

- Выделите объект **queue**. На панели **Свойства** измените **Вместимость** с 15 на 5 запросов.
- Здесь же установите **Разрешить вытеснение**.
- Для уничтожения потерянных запросов вследствие полного заполнения накопителя нужно добавить второй объект **sink**. Откройте в **Палитре Библиотеку моделирования процессов** и перетащите блок **sink** на диаграмму. При перетаскивании объект пытается автоматически соединиться с входами имеющимися на диаграмме объектами. Но это может вас не устраивать.
- Тогда соедините порт **outPreempted** объекта **queue** с входным портом **InPort** блока **sink1**. Чтобы соединить порты, сделайте двойной щелчок мышью по одному порту, например, **outPreempted**, затем последовательно Щёлкните в тех местах диаграммы, где вы хотите поместить точки изгиба соединителя.
- После двойного щелчка по второму порту вы увидите, что появится соединитель. Если выделить его мышью, то при правильном соединении портов конечные точки соединителя должны подсветиться зелеными точками. Если нет, то точки не были помещены точно внутрь портов, и их нужно будет туда передвинуть.



2. Запустите уточненную модель и наблюдайте за ее работой. Видно, что запросы при длине очереди в 5 запросов теряются, и ошибки при этом не возникает. Модель по ограничению ёмкости входного буфера и значениям других параметров соответствует постановке задачи.



Однако согласно постановке задачи требуется определить математическое ожидание времени обработки одного запроса и математическое ожидание вероятности обработки запросов.

Сбор статистики по показателям обработки запросов

Агент (заявка) являются базовым классом для всех заявок, которые создаются и работают с ресурсами в процессе, описанном вами с помощью диаграммы из объектов [Библиотеки моделирования процессов](#). Агент по существу является обычным Java классом с теми функциональными возможностями, которые необходимы и достаточны для обработки и отображения анимации заявки объектами [Библиотеки моделирования процессов](#). Эти функциональные возможности можно расширить добавлением дополнительных полей и методов и работой с ними из объектов диаграммы, описывающей моделируемый процесс.

Согласно постановке задачи нужно определять математическое ожидание времени и вероятности обработки запросов сервером.

Математическое ожидание или среднее время обработки одного запроса определяется как отношение суммарного времени обработки n запросов к их количеству, т.е. к n . Для определения суммарного времени нужно знать время обработки i -го запроса. Для этого введем дополнительные поля:

- **time_vxod** - время входа запроса в буфер сервера,
- **time_vixod** - время выхода запроса с сервера (входа в блок **sink**).

Тогда

$$\text{time_obrabotki} = \text{time_vixod} - \text{time_vxod}$$

Вероятность обработки запросов сервером определяется как отношение количества обработанных запросов к количеству всех поступивших запросов. Значит, нужно вести счет

запросов на выходе источника запросов и на выходе с сервера (входе в блок [sink](#)). Для этого также введем дополнительные поля:

- [col_vxod](#) - количество поступивших всего запросов,
- [col_vixod](#) - количество обработанных сервером запросов.

Тогда

$ver_obrabotki = col_vixod / col_vxod$

Замечание. Ничего необычного во введенных дополнительных полях нет. Это параметры реальных элементов потоков, в данном случае запросов. AnyLogic предоставляет возможность создавать запросы с теми параметрами, которые необходимы в модели.

Создание нестандартного Java класса

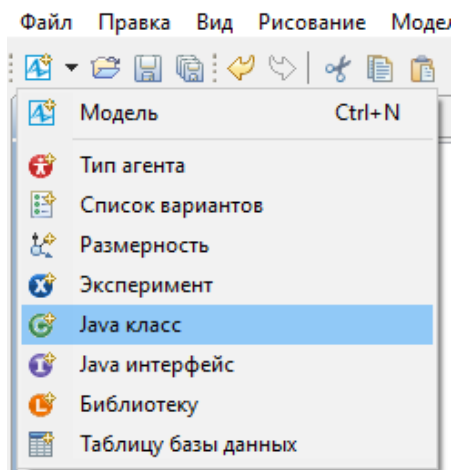
Для включения в запросы дополнительных полей необходимо создать нестандартный тип заявки. Это возможно двумя способами.



В учебной версии AnyLogic сохранение созданного Java класса невозможно, поэтому первый способ приводится в ознакомительных целях.

Первый способ предусматривает создание нового типа заявок Inquiry:

- В панели [Проект](#) щёлкните правой кнопкой мыши элемент модели верхнего уровня дерева и выберите из контекстного меню [Создать/Java класс](#).



- Появится диалоговое окно [Новый Java класс](#). В поле [Имя](#): введите имя нового класса [Inquiry](#).

- В поле [Базовый класс](#): выберите из выпадающего списка [Агент](#) в качестве базового класса. Щёлкните кнопку [Далее](#).

- Появится вторая страница Мастера создания Java класса.

Добавьте поля Java класса:

- `time_vxod` типа `double`,
- `time_vixod` типа `double`,
- `col_vxod` типа `int`,
- `col_vixod` типа `int`.

Типы полей выбираются из выпадающего списка. Начальные значения всех параметров, поскольку не указаны, по умолчанию будут установлены равными нулю.

- Оставьте выбранными флажки **Создать конструктор** и **Создать метод `toString()`**. Тогда у класса будут созданы сразу два конструктора: один, по умолчанию, без параметров, и второй, с параметрами, инициализирующими поля класса. Эти конструкторы используются объектами, создающими новые заявки, такие, как `Source`.

Новый Java класс

Поля класса

Добавьте поля Java класса

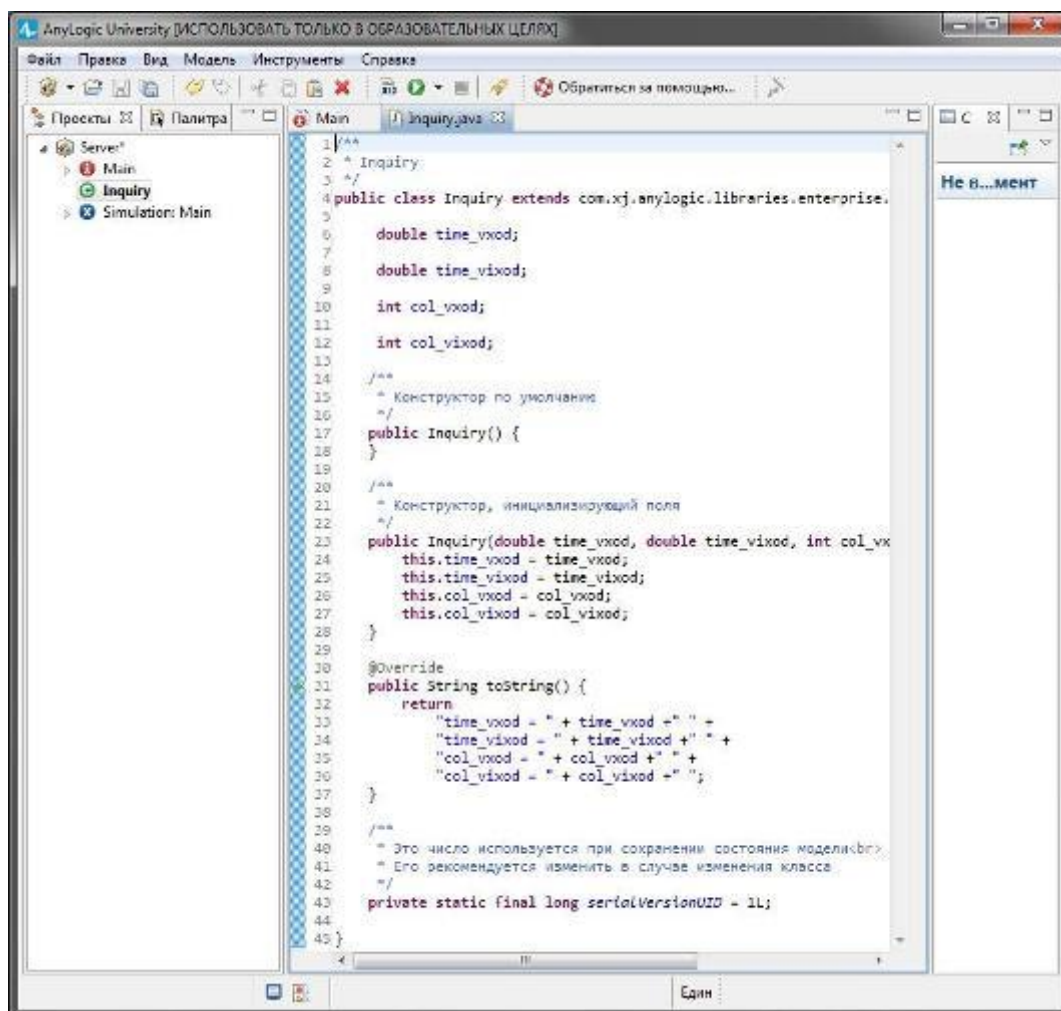
Имя	Тип	Доступ	Начальное значе...
time_vxod	double		
time_vixod	double		
col_vxod	int		
col_vixod	int		

☒ Создать конструктор

☒ Создать функцию `toString()`

< Назад Далее > Готово Отменить

- Щёлкните кнопку **Готово**. Вы увидите редактор кода, в котором будет показан автоматически созданный код вашего Java класса. Закройте редактор, щелкнув крестик в закладке рядом с его названием.



- Теперь нужно преобразовать Java класс в тип агента. Для этого щёлкните правой кнопкой мыши в панели **Проект** только что созданный Java класс и в контекстном меню выберите **Преобразовать Java класс в тип агента**.

- Появится окно с автоматически созданными параметрами нестандартного типа заявок **Inquiry**.

- Закройте окно, щелкнув крестик в закладке рядом с его названием.

- Если вы в учебной версии, то удалите всё, что касается только что созданного Java класса и заявки и воспользуйтесь вторым способом.

Перейдём к созданию непосредственно нестандартного типа заявки вторым способом.

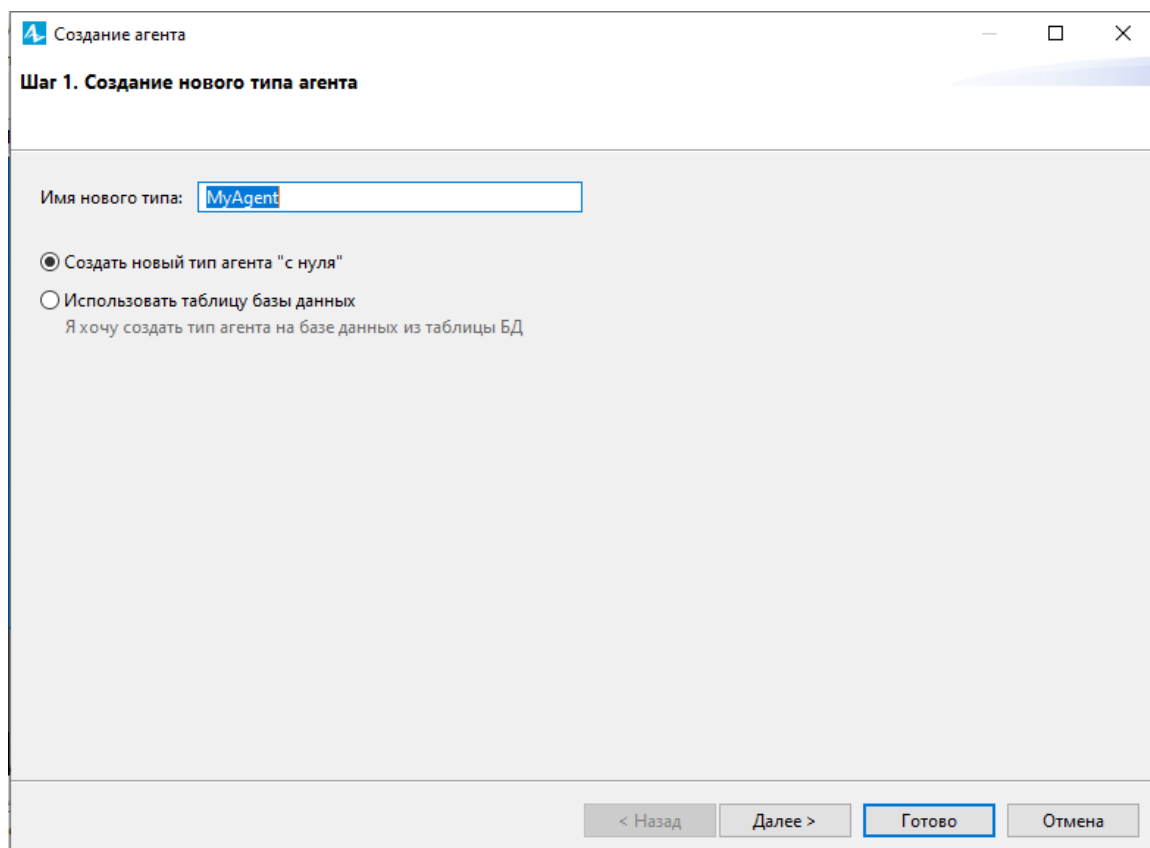
1. Создайте тип агента **Inquiry**.

- Откройте палитру **Библиотека моделирования процессов**.

- Перетащите элемент **Тип агента** в графический редактор.

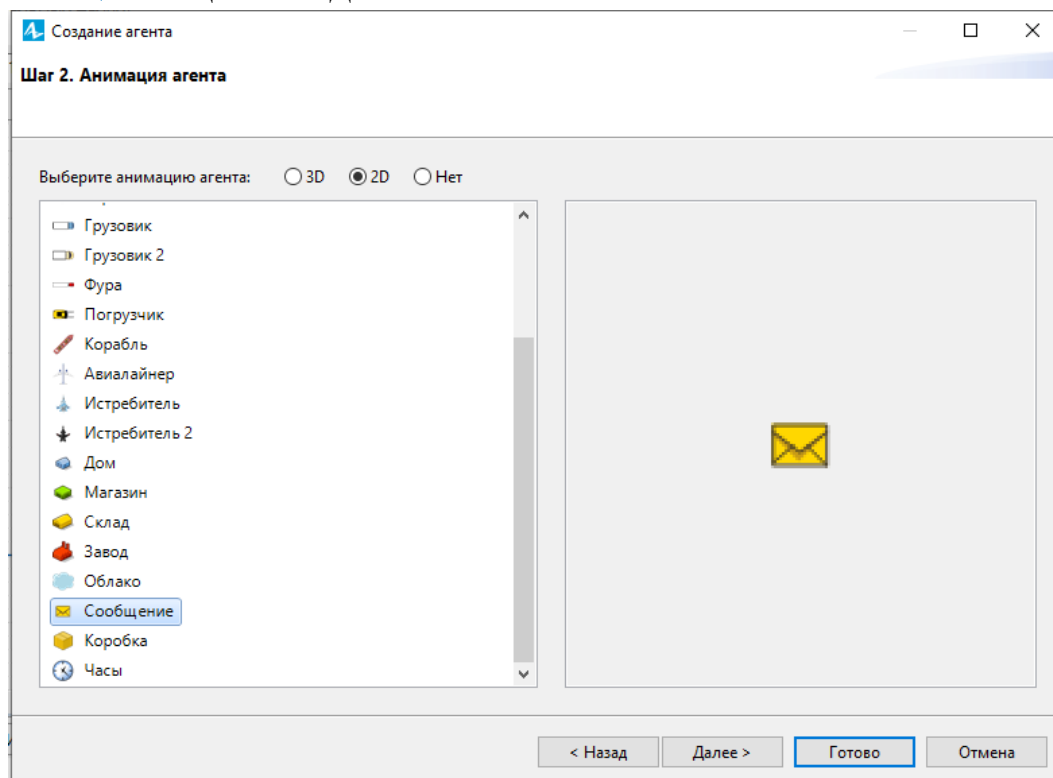
2. Появится диалоговое окно **Создание агента. Шаг 1. Создание нового типа агента**.

- В поле **Имя нового агента**: введите **Inquiry**. Щёлкните **Далее**.

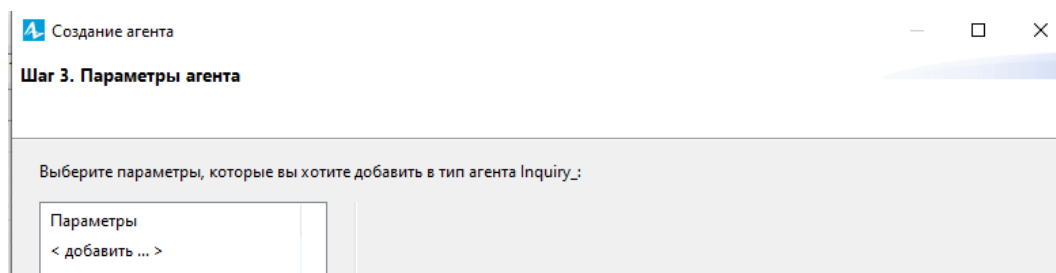


3. Появится диалоговое окно [Создание агента. Шаг 2. Анимация агента](#)

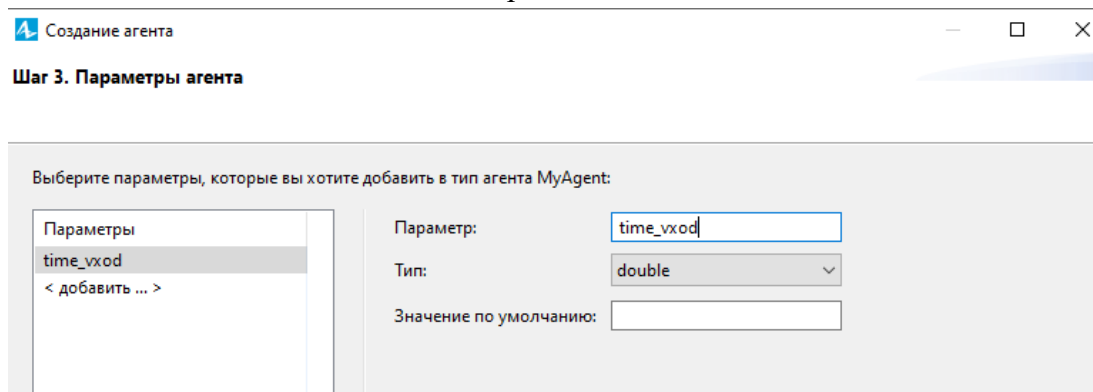
- Выберите анимацию агента: [установите 2D](#) и выберите из выпадающего списка, например, [Сообщение](#). Щёлкните Далее.



4. Появится диалоговое окно [Создание агента. Шаг 3. Параметры агента](#)



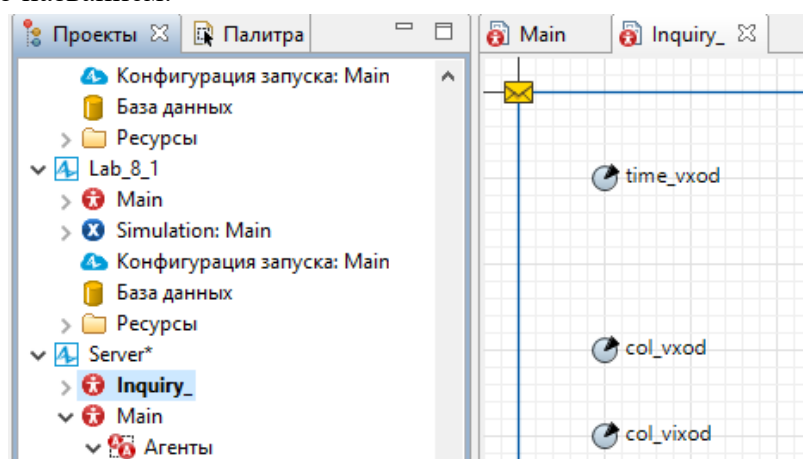
- Щёлкните <добавить...>. В поле **Параметр:** введите `time_vxod`
- Из выпадающего списка **Тип:** выберите `double`.



- Щёлкните третий раз <добавить...>. В поле **Параметр:** введите `col_vxod`.
- Из выпадающего списка **Тип:** оставьте `int`.
- Щёлкните третий раз <добавить...>. В поле **Параметр:** введите `col_vixod`.
- Из выпадающего списка **Тип:** оставьте `int`.

Так как в поле **Значение по умолчанию** мы не устанавливали никаких значений, то всем параметрам будет установлен `0`.

- Щёлкните кнопку **Готово**. Вы увидите окно, в котором будут показаны автоматически созданные параметры нестандартного типа заявок **Inquiry**. Закройте оно, щелкнув крестик в закладке рядом с его названием.

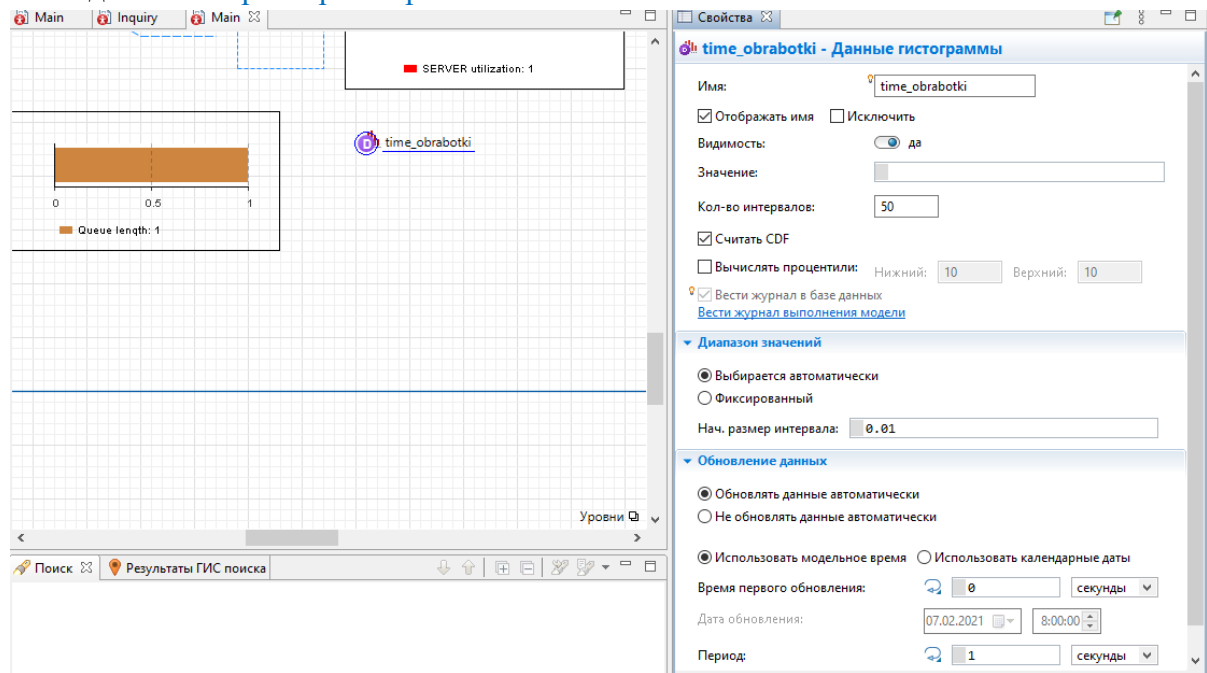


Добавление элементов статистики

1. Для сбора статистических данных о времени обработки запросов сервером необходимо добавить элемент статистики. Этот элемент будет запоминать значения времен для каждого запроса. На основе этого он предоставит пользователю стандартную статистическую информацию (среднее, минимальное, максимальное из измеренных значений, среднееквадратичное отклонение и т.д.).

- Перетащите элемент **Данные гистограммы** с палитры **Статистика** на диаграмму активного класса.
- Задайте свойства элемента:

- измените **Имя:** на `time_obrabotki`;
- сделайте **Кол-во интервалов:** равным **50**;
- задайте **Нач. размер интервала:** **0.01**.

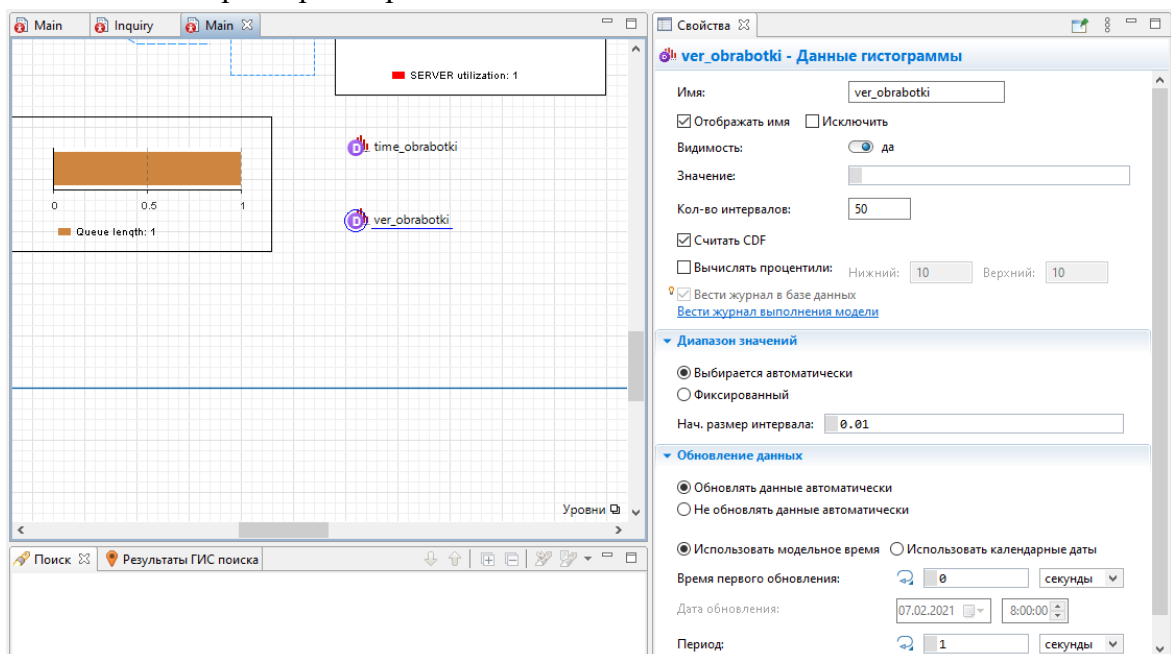


2. Добавьте еще элемент сбора статистики для определения вероятности обработки запросов.

- Перетащите элемент Данные гистограммы с палитры Статистика на диаграмму активного класса.

- Задайте свойства элемента:

- измените **Имя:** на `ver_obrabotki`;
- сделайте **Кол-во интервалов:** равным **50**;
- задайте **Нач. размер интервала:** **0.01**.



Изменение свойств объектов диаграммы

Чтобы создавать заявки нестандартного типа, как в нашем случае **Inquiry**, вам нужно поместить вызов конструктора этого типа в поле **Новый агент** объекта **source**. Но, несмотря на то, что заявки в потоке теперь и будут типа **Inquiry**, остальные объекты диаграммы будут продолжать их считать заявками типа **Агент**.

Поэтому они не позволяют явно обращаться к дополнительным полям класса **Inquiry**. Чтобы разрешить доступ к полям вашего нестандартного типа заявки в коде динамических параметров объектов потоковой диаграммы, вам нужно указать имя нестандартного типа заявки в качестве **Типа агента** этого объекта. В нашей потоковой диаграмме с учётом блока **source** всего пять объектов.

1. Измените свойства объекта **source** :

- введите **Inquiry** в поле **Тип агента**:. Это позволит напрямую обращаться к полям типа агента **Inquiry** в коде динамических параметров этого объекта;
- выберите из выпадающего списка **Inquiry** в поле **Новый агент**:. Теперь этот объект будет создавать заявки нашего типа **Inquiry**;
- введите **agent.time_vxod=time()**; в поле **Действия При выходе**:. Код будет сохранять время создания заявки-запроса в параметре **time_vxod** нашего типа заявки **Inquiry**.



Функция **time()** возвращает текущее значение модельного времени.

2. Измените свойства объекта **queue**:

- введите **Inquiry** в поле **Тип агента**..

3. Измените свойства объекта **delay**:

- введите **Inquiry** в поле **Тип агента**..

4. Измените свойства объекта **sink1**:

- введите **Inquiry** в поле **Тип агента**..

5. Измените свойства объекта **sink**:

- введите **Inquiry** в поле **Тип агента**..;

- введите в поле **Действие при входе** следующие коды:

- o **time_obrabotki.add(time()-agent.time_vxod);**



Этот код добавляет время обработки одного запроса в объект сбора данных гистограммы `time_obrabotki`. Данное время определяется как разность между текущим модельным временем `time()` и временем входа запроса в модель. `add` - встроенная функция добавления элемента в массив.

- `agent.col_vixod=sink.count();`
- `agent.col_vxod=source.count();`



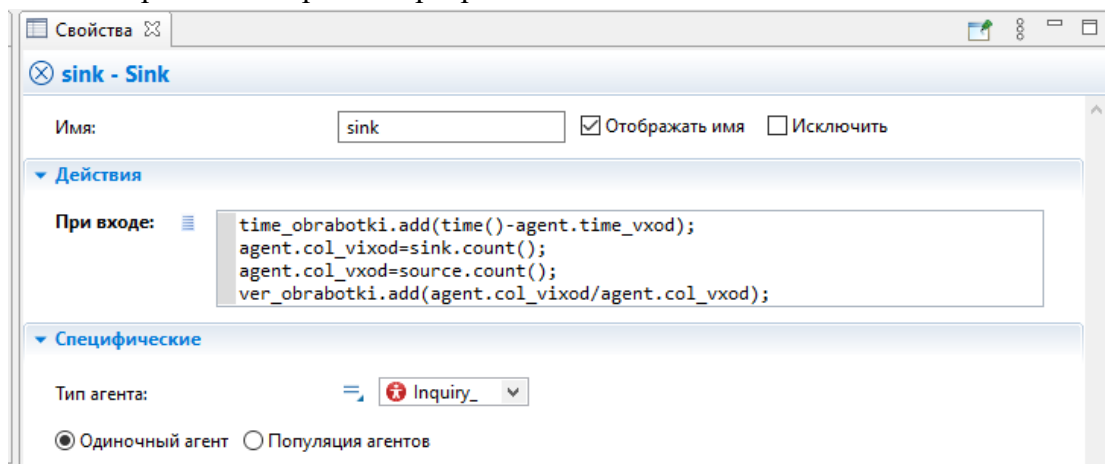
Эти коды заносят количество запросов, вошедших в блок `sink` и вышедших из блока `source` соответственно. `count()` - встроенная функция этих блоков, возвращает количество вошедших в блок `sink` и количество вышедших из блока `source` заявок.

- `ver_obrabotki.add(agent.col_vixod/ agent.col_vxod);`

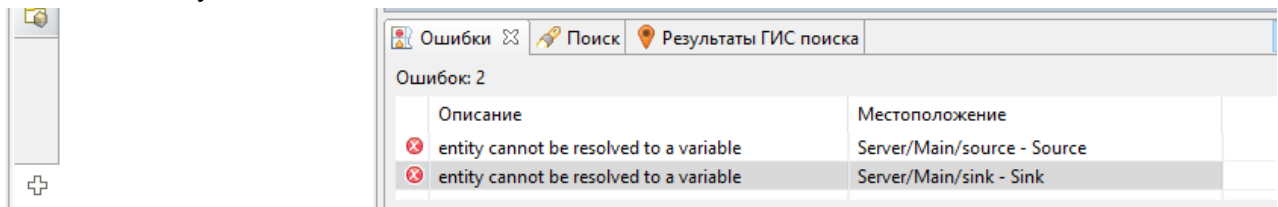


Этот код добавляет относительную долю обработанных запросов в объект сбора данных гистограммы `ver_obrabotki` при поступлении каждого обработанного запроса в блок `sink`.

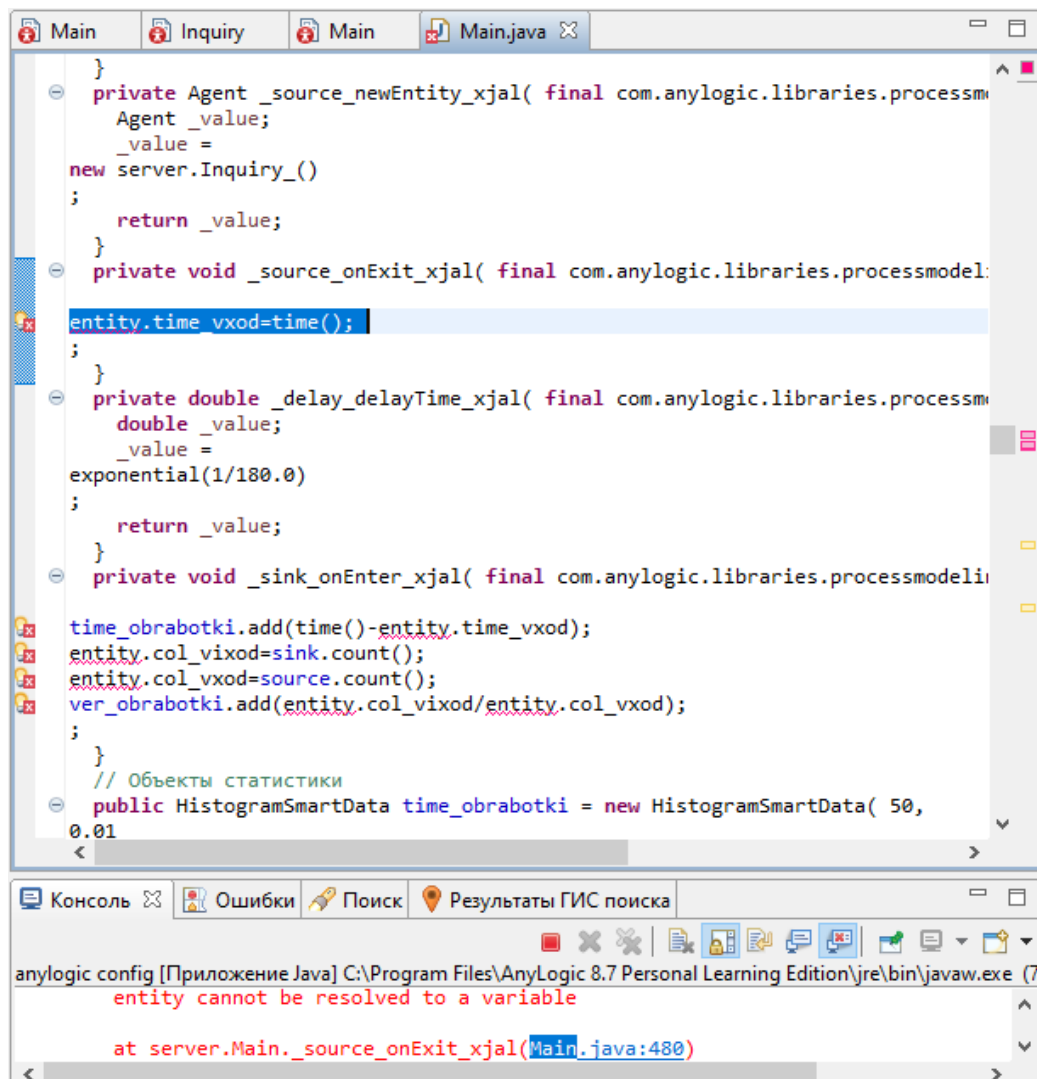
На основе множества таких относительных долей определяется математическое ожидание вероятности обработки запросов сервером:



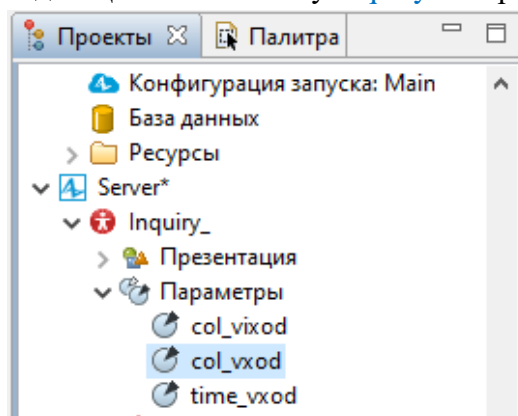
6. Запустите модель. Появится сообщение об ошибке.



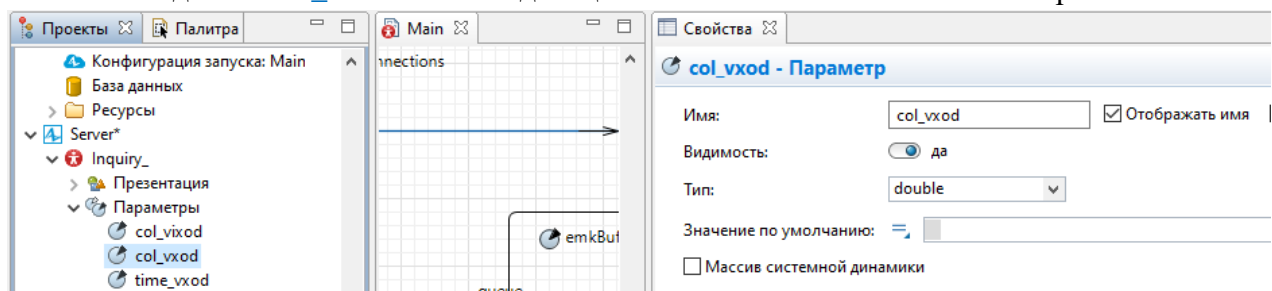
- Щёлкните выделенный [Java код](#) в панели [Консоль](#) например, [Main.java.480](#) . Появится код с выделенными ошибками:



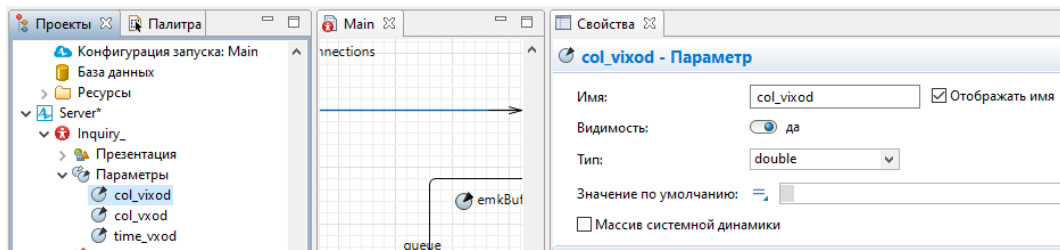
- Мы установили ранее тип `int` для `col_vxod` и `col_vixod`. Изменим этот тип на `double`:
 - В поле **Проект** дважды щёлкните кнопку **Inquiry**. Откроется окно **Inquiry**.



- Выделите `col_vxod`. Из выпадающего списка **Тип:** вместо `int` выберите `double`



- Выделите `col_vixod`. Из выпадающего списка **Тип:** вместо `int` выберите `double`.

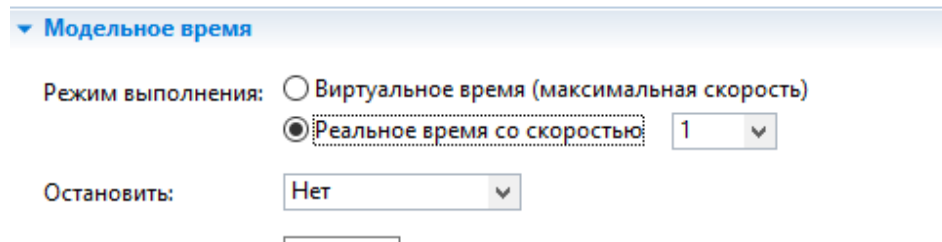


Итак, все условия постановки задачи выполнены.

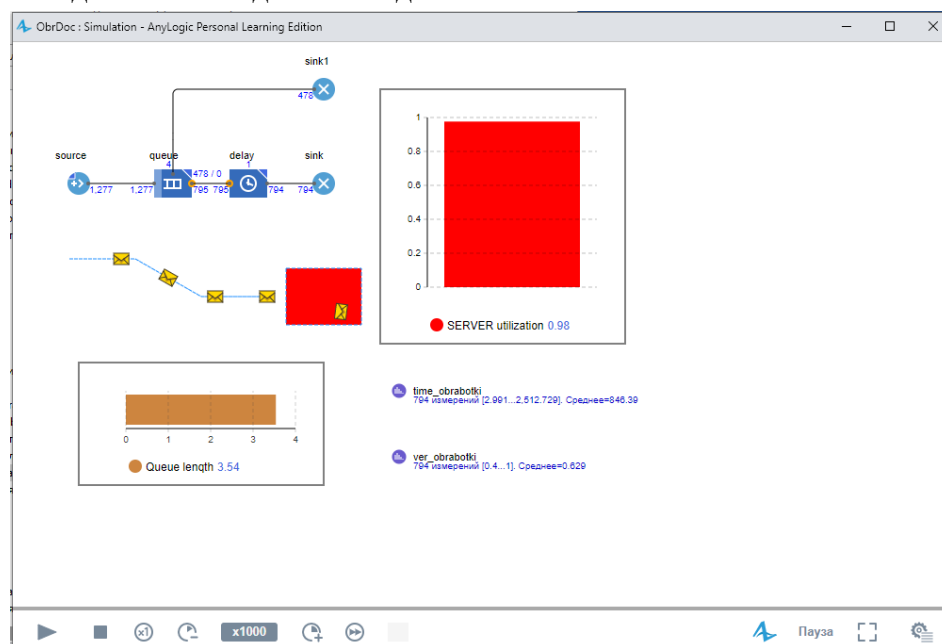
Запуск модели

- Чтобы наблюдать за работой модели, установите, что время остановки модели не задано и

Режим выполнения: Реальное время со скоростью - 1.



- Запустите модель. Далее изменяя скорость работы модели в окне симуляции наблюдайте за поведением модели:



Добавление параметров и элементов управления

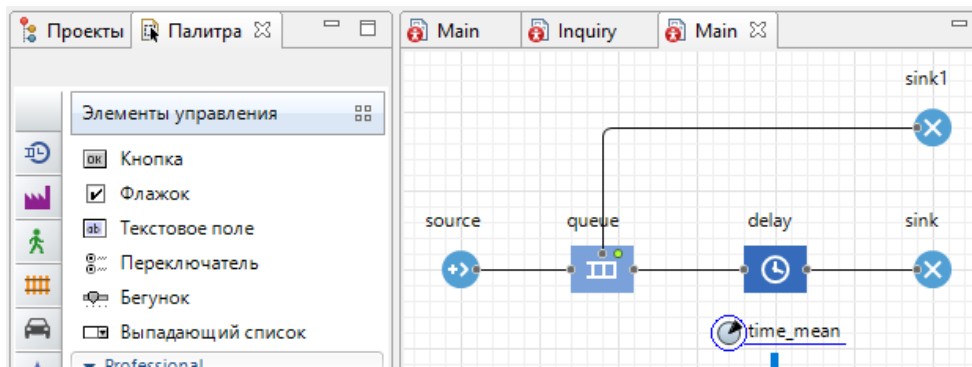


Активный объект может иметь параметры. Параметры обычно используются для задания статических характеристик объекта. Но значения параметров при необходимости можно изменять во время работы модели. Для этого нужно написать код обработчика события, то есть действий, которые должны выполняться при изменении значения параметра.

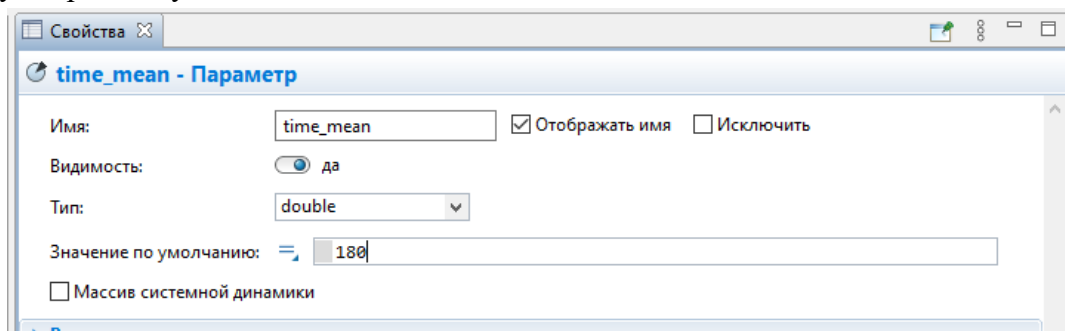
1. Создайте параметр **time_mean** объекта **delay**.

- В **Палитре** выделите **Системная динамика**.

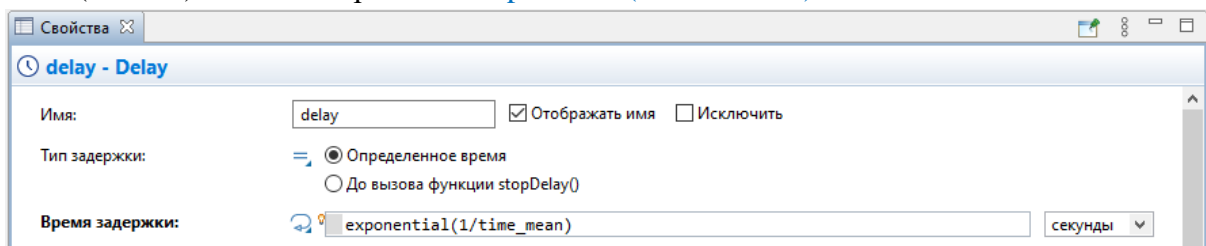
- Перетащите элемент **Параметр** на диаграмму класса **Main** и разместите ниже объекта **delay**, чтобы было видно, к какому объекту относится параметр.



- Перейдите на панель **Свойства**
- В поле **Имя** введите имя параметра **time_mean** (среднее время). По этому имени параметр будет доступен из кода.
- Задайте тип параметра **double**.
- В поле **Значение по умолчанию** установите **180**. Если значение не задано явно, по правилам Java оно будет равно нулю.

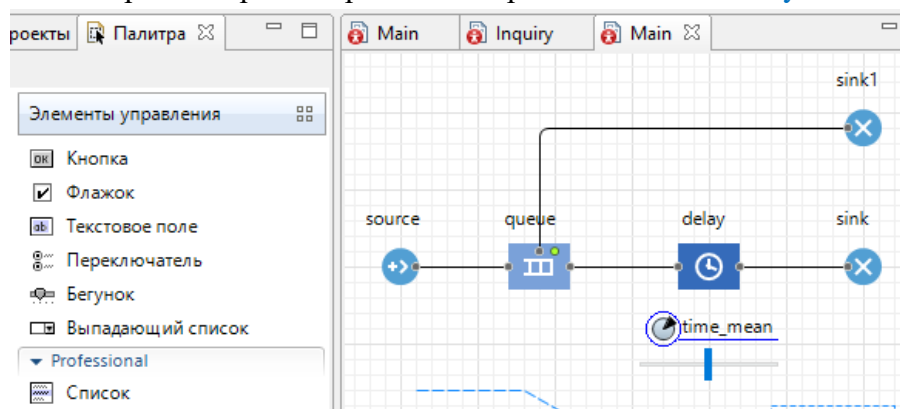


2. Выделите объект **delay**. На панели **Свойства** в поле **Время задержки** вместо выражения $\text{exponential}(1/180.0)$ введите выражение $\text{exponential}(1/\text{time_mean})$.

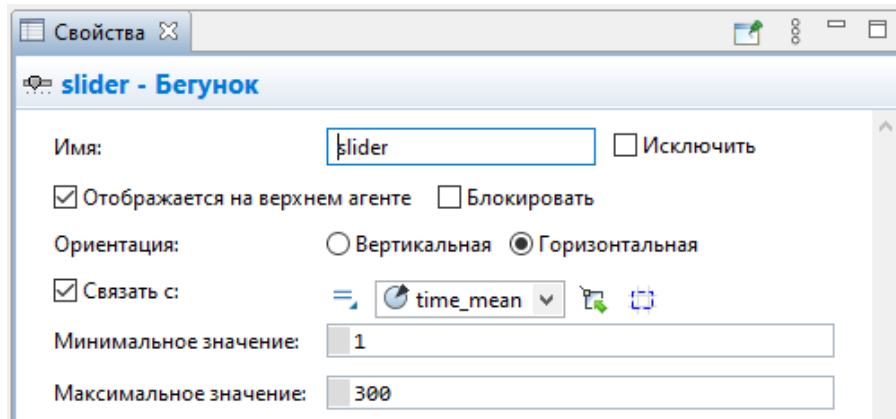


3. Для изменения среднего времени обработки запросов **time_mean** в ходе моделирования используйте элемент управления - бегунок.

- Откройте палитру **Элементы управления** и перетащите элемент **Бегунок** из палитры на диаграмму класса **Main**.
- Поместите бегунок под параметром **time_mean**, чтобы было понятно, что с помощью этого бегунка будет меняться среднее время обработки запросов объектом **delay**.



- Чтобы варьировать среднее время от 1 до 300 введите 1 в поле **Минимальное значение:**, а 300 - в поле **Максимальное значение.**
- Установите флажок **Связать с:** и в активизированное поле введите **time_mean**.



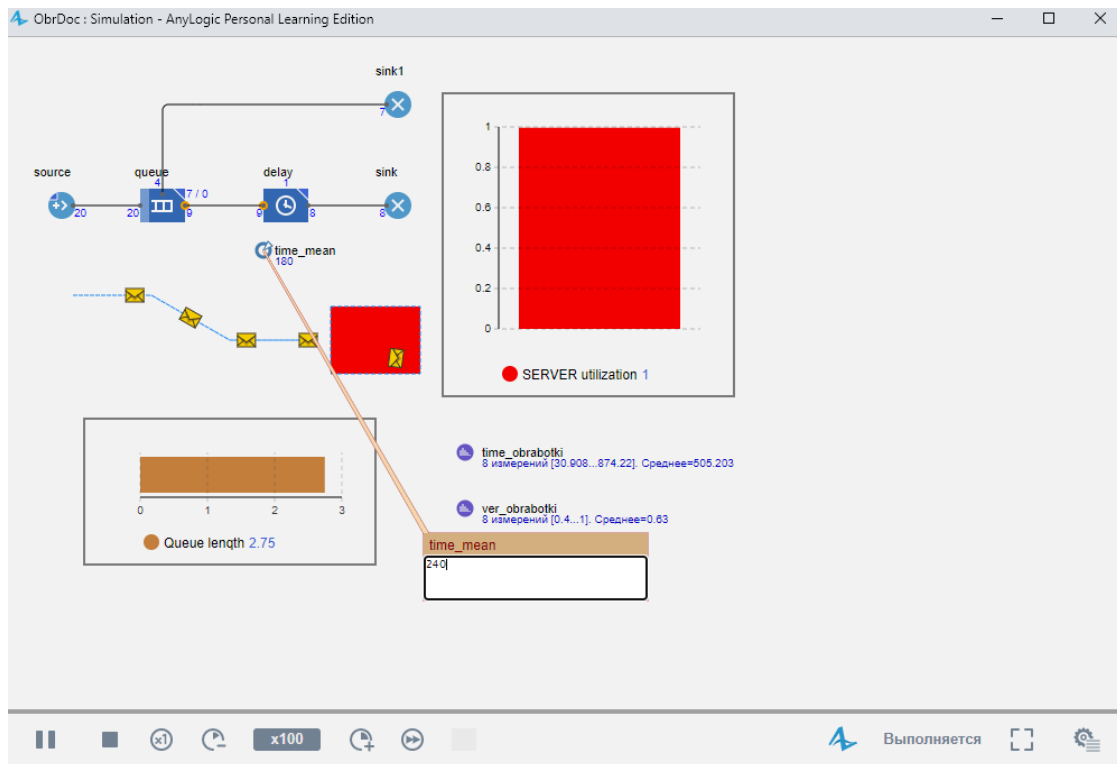
4. Запустите модель. Теперь вы можете изменять в процессе моделирования среднее время обработки запросов с помощью бегунка. Можете также командой Приостановить работу модели, изменить значения параметров, а затем продолжить моделирование.

5. Остановите модель и перейдите на диаграмму класса Main.

Но согласитесь, что какие параметры модели нужно будет менять, и в каких интервалах, заранее определить затруднительно. Также при использовании элемента Бегунок существуют трудности точного установления значения характеристики, так как невозможно предусмотреть нужный масштаб или цену деления Бегунка.

6. Существует и другой способ изменения свойств объектов во время выполнения модели: нужно щёлкнуть по элементу, войти в режим редактирования и ввести новое значение в одной из закладок всплывающего окна инспекта. Поэтому заранее не нужно продумывать, значения каких параметров планируется изменять, и не добавлять специальные элементы управления (например, бегунки). Изменение значения в окне инспекта поддерживается для следующих элементов: простая переменная; параметр; накопитель. И для следующих типов: численные; логический (boolean); текстовый (String).

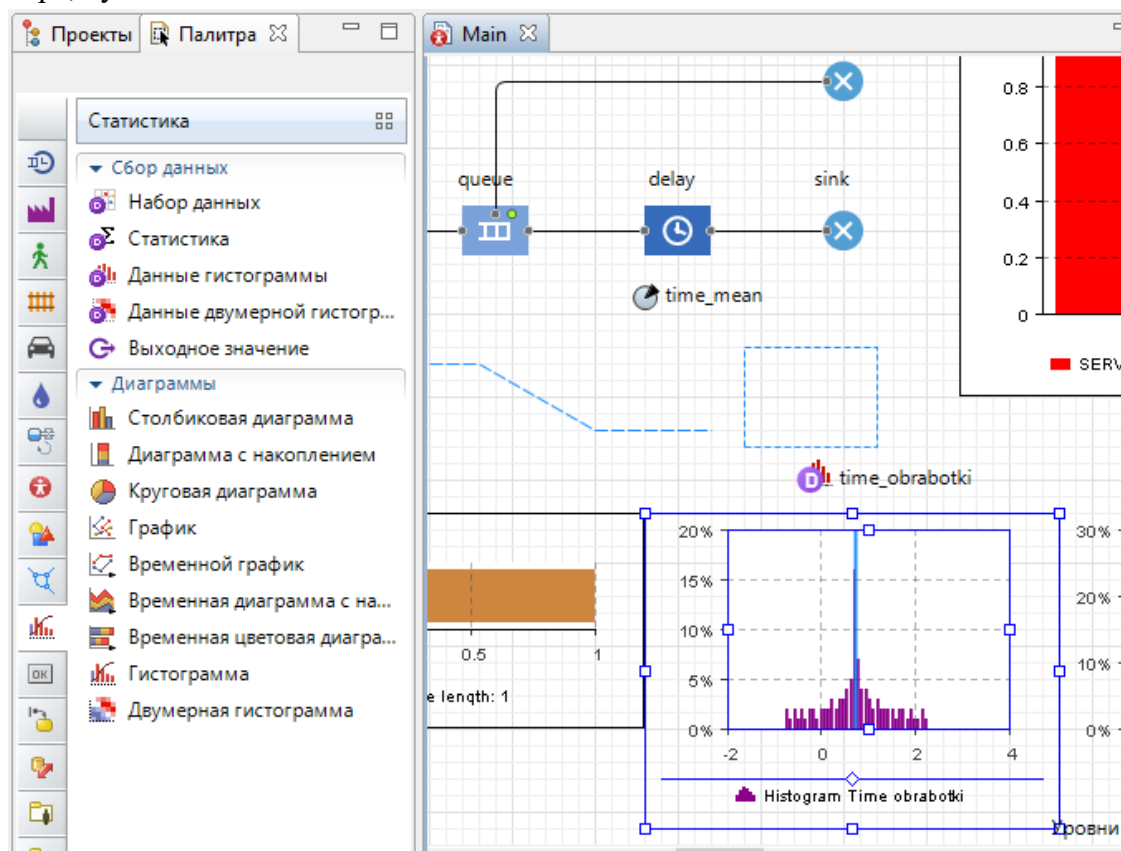
- Удалите элемент **Бегунок** для **Параметра** **time_mean**.
- Запустите модель и приостановите её.
- Щёлкните по значку **Параметра** **time_mean**.
- Перейдите в режим редактирования.
- Введите новое значение: 240.0.
- Закройте инспект. Рядом с элементом Параметр вы увидите введённое вами значение 240.0.
- Запустите модель с новым свойством объекта **delay**.



Добавление гистограмм

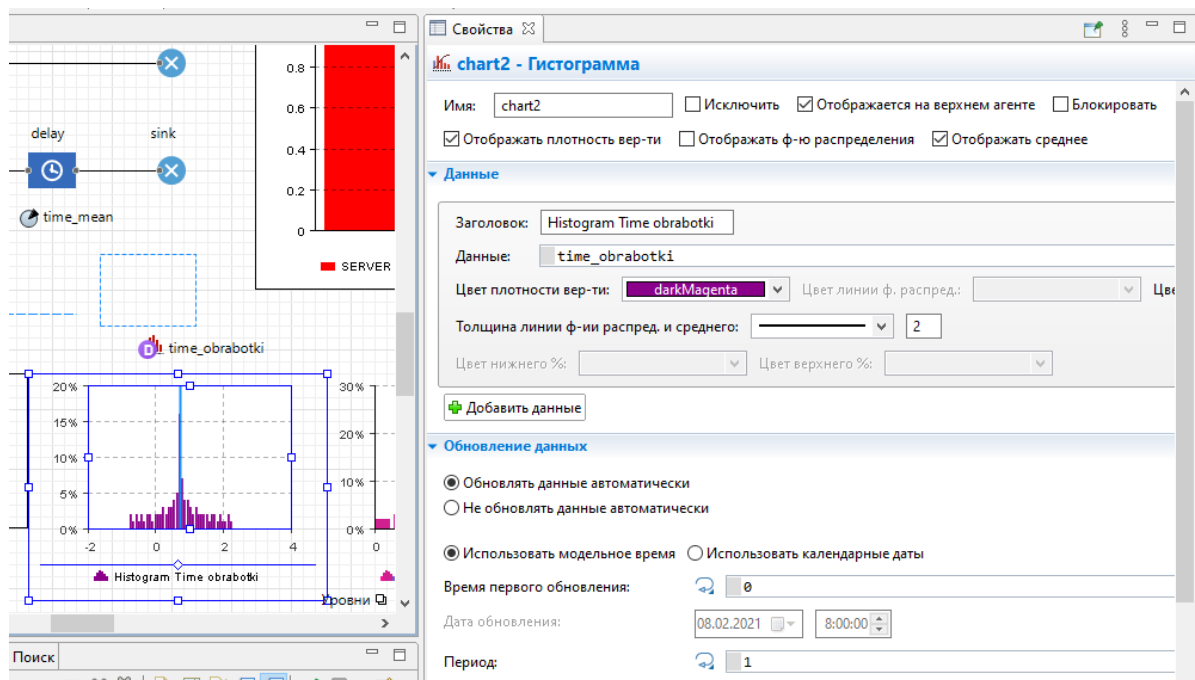
1. Добавьте на диаграмму нашего потока гистограмму, которая будет отображать собранную временную статистику:

- Перетащите элемент **Гистограмма** из палитры **Статистика** в то место графического редактора, куда хотите ее поместить.



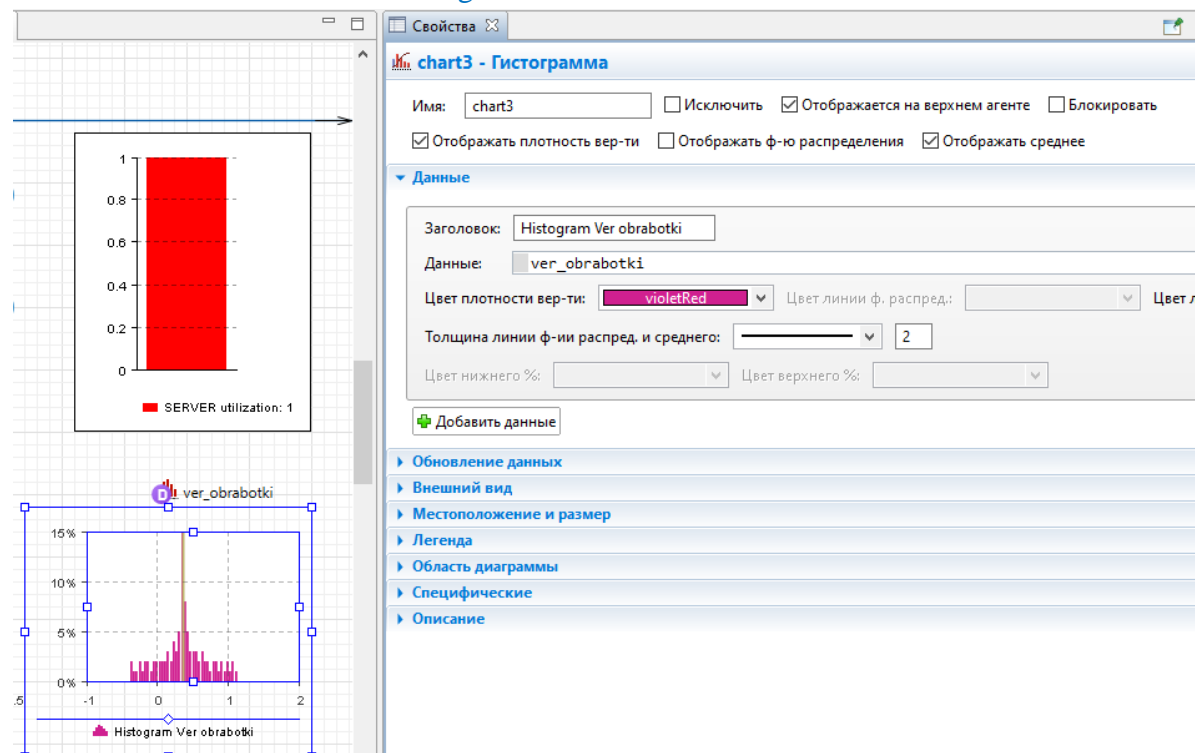
- Укажите, какой элемент сбора данных хранит данные, которые вы хотите отображать на гистограмме: введите в поле **Данные** имя соответствующего элемента: **time_obrabotki**. Установите **Отображать среднее**.

- В поле **Заголовок**: введите **Histogram Time obrabotki**.

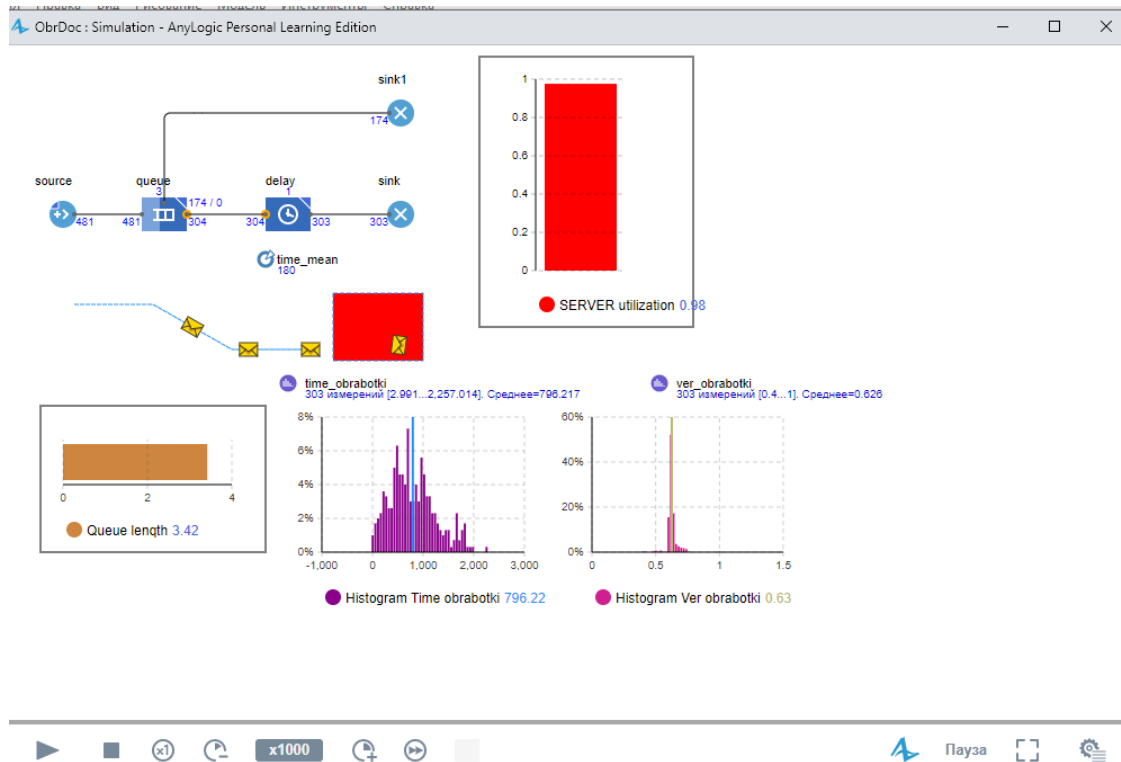


2. Добавьте на диаграмму нашего потока гистограмму, которая будет отображать собранную вероятностную статистику:

- Перетащите элемент **Гистограмма** из палитры **Статистика**.
- Введите в поле **Данные** имя элемента: **ver_obrabotki**. Установите **Отображать среднее**.
- В поле **Заголовок**: введите **Histogram Ver obrabotki**.



3. Запустите модель.



Доработка модели согласно условию задачи



В созданной модели, с целью упрощения процесса построения, время обработки запросов сервером было принято распределённым по показательному (экспоненциальному) закону со средним значением $T2 = 3$ мин. (см. п. Уяснение задачи моделирования)

Однако в модели время обработки поступающих запросов зависит от производительности сервера $Q = 6 \cdot 10^5$ оп/с и вычислительной сложности запросов, распределенной по нормальному закону с математическим ожиданием $S1 = 6 \cdot 10^7$ оп и среднеквадратическим отклонением $S2 = 2 \cdot 10^5$ оп.

Кроме того, в модели определяется среднее количество запросов, обработанных за время моделирования 3600 с (1 ч).

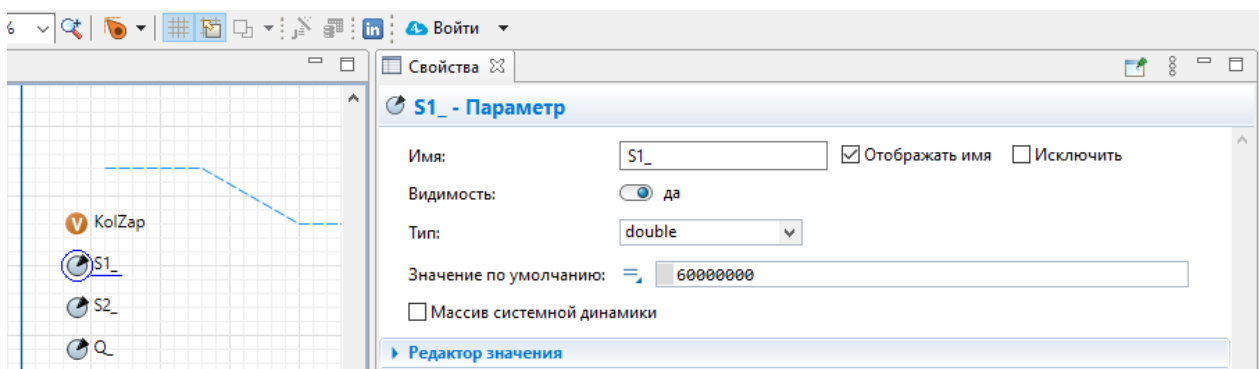
Внесите в модель изменения для аналогичного расчёта времени обработки запросов:

1. Удалите элемент **Параметр** с именем **time_mean** элемент **Бегунок** для элемента **queue**.

2. Из палитры **Агент** перетащите три элемента **Параметр** на диаграмму класса Main:

- В поле **Имя** каждого из элементов введите **S1_**, **S2_** и **Q** соответственно. Выберите Тип **double**.

- В поле **Значение по умолчанию** каждого из элементов введите **60000000**, **200000** и **600000** соответственно.



3. Перетащите элемент **Переменная**.

- В поле **Имя** укажите **KolZap**.

4. Выделите объект **delay**.

- В поле **Время задержки** вместо $\text{exponential}(1/\text{time_mean})$ введите: $(\text{normal}(S2_, S1_))/Q_$

Свойства

delay - Delay

Имя: ☒ Отображать имя ☐ Исключить

Тип задержки: ☒ Определенное время
☐ До вызова функции stopDelay()

Время задержки:

Вместимость:

Максимальная вместимость:

5. Выделите объект **sink**.

- В поле **Действие при входе** к имеющемуся там коду добавьте код:

KolZap= round(sink.in.count()/9604.0)

Свойства

sink - Sink

Имя: ☒ Отображать имя

☐ Исключить

Действия

При входе:

```
time_obrabotki.add(time()-agent.time_vxod);
agent.col_vixod=sink.count();
agent.col_vxod=source.count();
ver_obrabotki.add(agent.col_vixod/agent.col_vxod);
KolZap= round(sink.in.count()/9604.0);
```

Установка времени моделирования

Для получения результатов моделирования с доверительной вероятностью $\alpha=0,95$ и точностью $\epsilon=0,01$ нужно выполнить 9604 прогонов модели (см. п.п. Постановка задачи и Уяснение задачи моделирования).

Увеличим время моделирования в 9604 раз. А так как статистические данные о количестве обработанных запросов собираются за всё время моделирования, увеличенное в 9604 раз, то для получения среднего значения это количество нужно разделить на 9604, что и предусмотрено в коде.

Показатели моделируемой системы нужно определить в течение 3600 с (1 ч), поэтому время моделирования в AnyLogic составит $3600 \times 9604 = 34574400$ единиц модельного времени.

- В панели **Проект** выделите **Simulation**. На странице **Модельное время** в поле **Установить** выберите **В заданное время**.

- В поле **Конечное время** установите **34574400**.

Свойства ✖

Simulation - Простой эксперимент

Имя: ☐ Исключить

Агент верхнего уровня:

Максимальный размер памяти: M6

☒ Пропустить экран эксперимента и запустить модель

▼ Модельное время

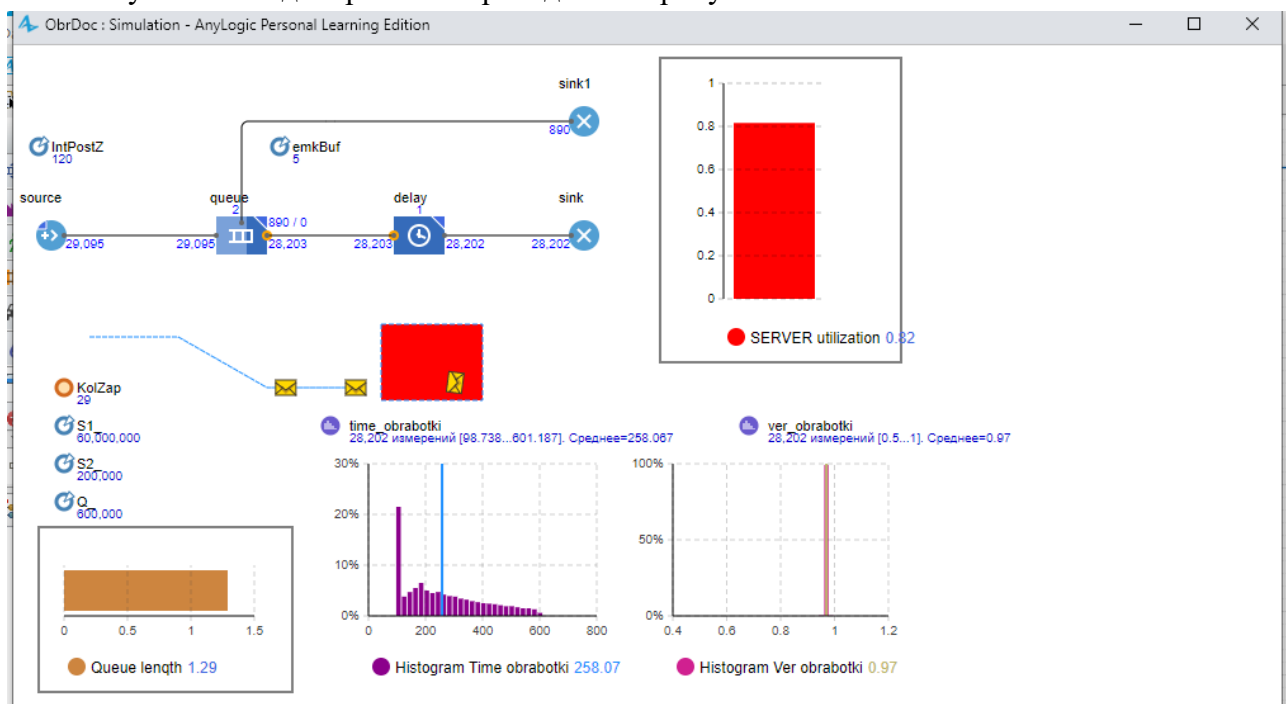
Режим выполнения: ☒ Виртуальное время (максимальная скорость)
☐ Реальное время со скоростью

Остановить:

Начальное время: Конечное время:

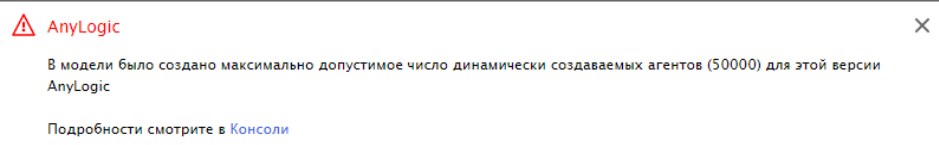
Начальная дата: Конечная дата:

- Запустите модель и дождитесь окончания моделирования.
 Результаты моделирования приведены на рисунке ниже:



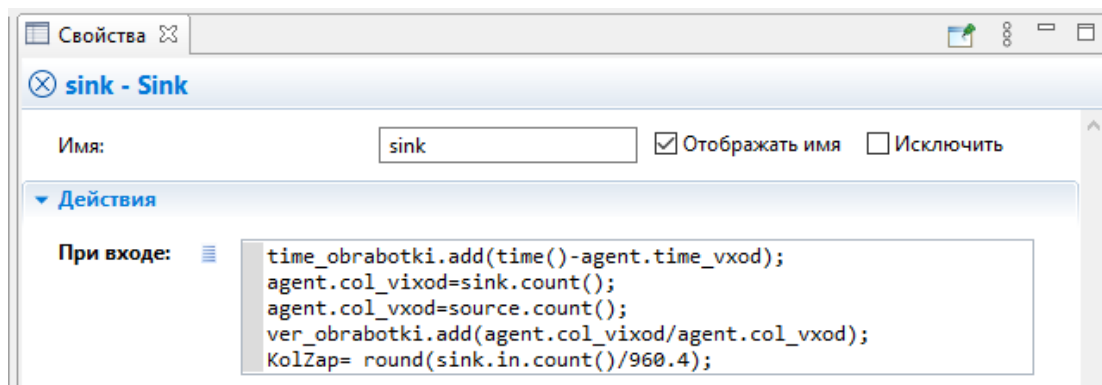
Внимание. Учебная версия AnyLogic не позволяет при симуляции работы модели производить более 50000 агентов.

Поэтому, если возникнет такая ошибка:

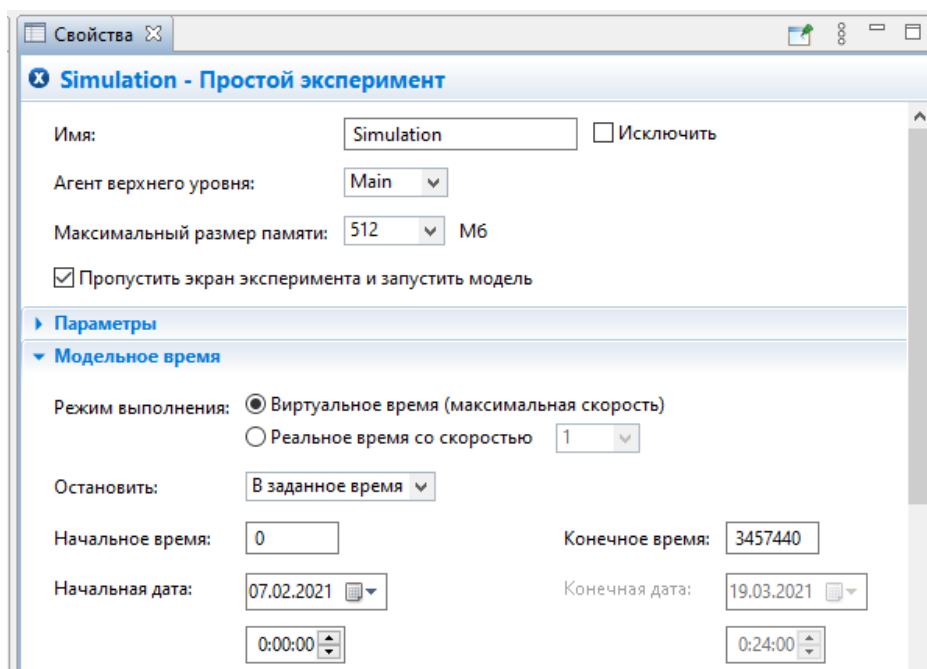


скорректируйте время моделирования, например, разделив количество прогонов на 10: $9604/10=960.4$.

Данное значение необходимо внести в объект **sink**, где производится расчет количества запросов: `KolZap= round(sink.in.count()/960.40):`



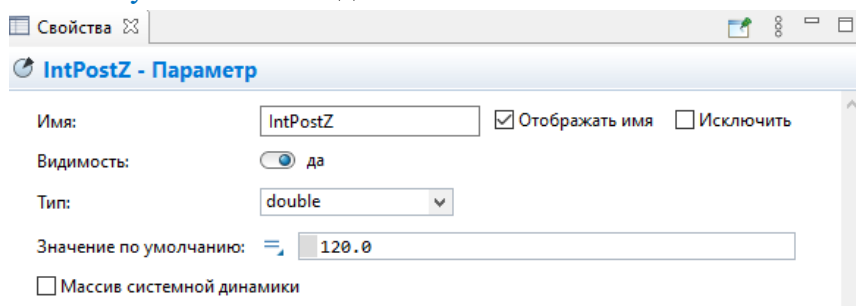
Учтите при этом, что **модельное время** тоже изменится: **3457440** единиц модельного времени:



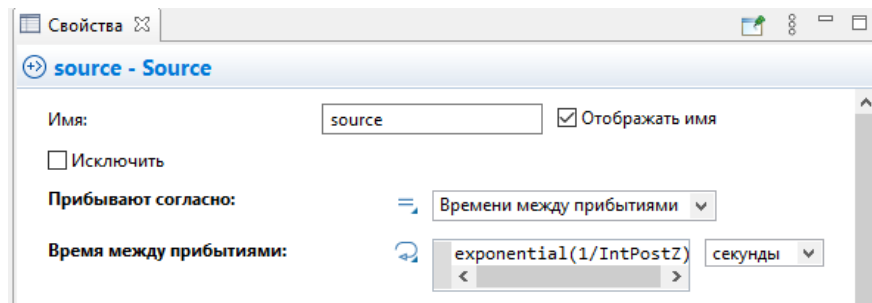
Проведение экспериментов с моделью

1. Чтобы проводить эксперименты с моделью целесообразно вынести параметры: ёмкость входного буфера (emkBuf) и среднее время поступления запросов (IntPostZ) на диаграмму класса Main, чтобы наглядно отследить их влияние на результат работы модели.

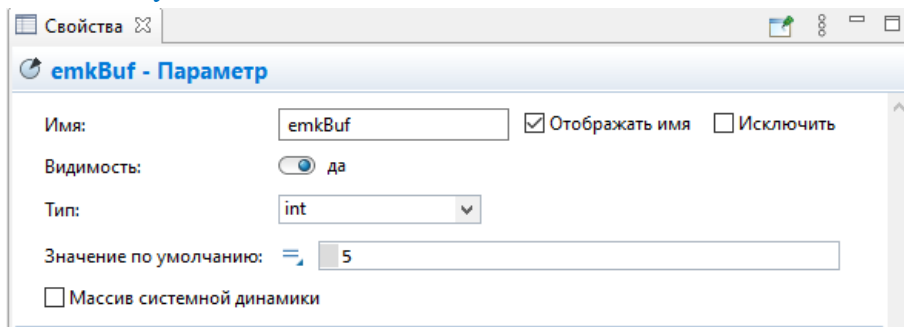
- Из палитры **Основная** перетащите два элемента **Параметр** на диаграмму агента **Main**.
- Выделите первый элемент **Параметр**. В поле **Имя**: первого параметра введите **IntPostZ** - среднее время поступления запросов для обработки на сервере. Оставьте тип **double**.
- В поле **Значение по умолчанию** введите **120**.



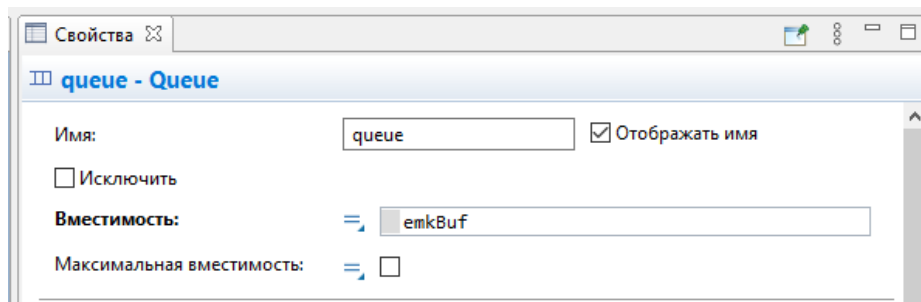
- Выделите объект **source**.
- В поле **Время между прибытиями** вместо 120.0 введите **IntPostZ**. Теперь вам при изменении среднего времени поступления запросов не придётся искать нужный код в свойствах объекта модели.



- Выделите второй элемент **Параметр**. В поле **Имя:** введите **emkBuf** - ёмкость в сообщениях входного буфера сервера.
- Установите тип **int**.
- В поле **Значение по умолчанию** введите **5**.



- Выделите объект **queue**.
- В поле **Вместимость** введите **emkBuf**.



2. Измените параметр Q (производительность сервера), увеличив исходное значение в 1,5-2 раза. Сделайте вывод как изменятся показатели работы сервера.
3. Измените параметр IntPostZ (интервал между поступлениями запросов), уменьшив и увеличив исходное значение в 2-3 раза. Сделайте вывод как изменятся показатели работы сервера.
4. Измените параметр emkBuf (ёмкость входного буфера), увеличив исходное значение в 2-3 раза. Сделайте вывод как изменятся показатели работы сервера.

Интерпретация результатов моделирования

Результаты экспериментов приведены в таблице 7.1

Таблица 7.1 - Показатели обработки запросов сервером

Показатели	Результаты в модели AnyLogic
Эксперимент 1: $IntPostZ = 120$, $emkBuf = 5$	
Количество обработанных запросов	29
Вероятность обработки запросов	0,97
Среднее время обработки одного запроса	255,727
Средняя длина очереди запросов к серверу	1,261
Коэффициент использования сервера	0,81
Эксперимент 2: $IntPostZ = 120$, $emkBuf = 10$	
Количество обработанных запросов	30
Вероятность обработки запросов	0,995
Среднее время обработки одного запроса	325,017
Средняя длина очереди запросов к серверу	1,87
Коэффициент использования сервера	0,831
Эксперимент 3: $IntPostZ = 40$, $emkBuf = 5$	
Количество обработанных запросов	36
Вероятность обработки запросов	0,4
Среднее время обработки одного запроса	554,983
Средняя длина очереди запросов к серверу	4,55
Коэффициент использования сервера	1
Эксперимент 4: $IntPostZ = 40$, $emkBuf = 10$	
Количество обработанных запросов	36
Вероятность обработки запросов	0,4
Среднее время обработки одного запроса	1054,907
Средняя длина очереди запросов к серверу	9,549
Коэффициент использования сервера	1
Эксперимент 5: $IntPostZ = 40$, $emkBuf = 10$, $Q_- = 1000000$	
Количество обработанных запросов	60
Вероятность обработки запросов	0,667
Среднее время обработки одного запроса	591,129
Средняя длина очереди запросов к серверу	8,852
Коэффициент использования сервера	1
Эксперимент 6: $IntPostZ = 40$, $emkBuf = 15$, $Q_- = 2000000$	
Количество обработанных запросов	90
Вероятность обработки запросов	1
Среднее время обработки одного запроса	75,096
Средняя длина очереди запросов к серверу	1,128
Коэффициент использования сервера	0,751

1. Из экспериментов 1 и 2 следует, что при увеличении ёмкости ($emkBuf$) входного буфера в два раза вероятность обработки запросов увеличивается незначительно на 0,025, то есть количество обработанных запросов практически одно и тоже. Среднее время обработки одного запроса возрастает в 1,27 раза вследствие увеличения длины очереди в 1,48 раза.

2. **Увеличение интенсивности поступления запросов ($1/\text{IntPostZ}$)** в три раза при увеличении ёмкости входного буфера в два раза (эксперименты 3 и 4) также не даёт существенного увеличения количества обработанных запросов: 36 вместо 30.
3. При этом уменьшается вероятность обработки запросов в 2,5 раза, а время обработки одного запроса возрастает в 3,25 раза. Возросла и средняя длина очереди запросов к серверу
4. Коэффициент использования сервера равен 1. Из этого следует, что добиться увеличения вероятности и количества обработанных запросов можно только увеличением производительности сервера.
5. В экспериментах 5 и 6 увеличена производительность сервера до 1000000 и 2000000 оп/с соответственно.
6. По сравнению с экспериментом 1 количество обработанных запросов в эксперименте 5 увеличилось в 2 раза, а в эксперименте 6 - в 3 раза (вероятность обработки запросов равна 1). Среднее время обработки одного запроса всё равно примерно в 2 раза больше в эксперименте 5, а в эксперименте 6 - в 3,4 раза меньше. Можно полагать, что такая разница в среднем времени обработки одного запроса вызвана тем, что в эксперименте 5 средняя длина очереди запросов к серверу 8,852, а в эксперименте 6 - 1,128, то есть в 7,85 раза меньше.
7. Коэффициент использования сервера в эксперименте 6 равен 0,751. Из этого следует, что дальнейшее увеличение производительности сервера приведёт к уменьшению длины очереди и среднего времени обработки одного запроса.
8. Машинное время выполнения модели в AnyLogic примерно 70...90 с.

ИНДИВИДУАЛЬНЫЕ ИСХОДНЫЕ ДАННЫЕ

Таблица 7.2 – Индивидуальные исходные данные

Номер варианта	Средний интервал поступления запросов IntPostZ , мин	Ёмкость входного буфера, emkBuf	Доверительная вероятность α	Вероятность обработки запросов сервером ρ	Точность, вычислений ε	Производительность сервера Q , оп/сек	Математическое ожидание $S1$, оп	Среднеквадратическое отклонение $S2$, оп
1	1	5	0,90106	0,5	0,01	600000	557280	34688
2	2	6	0,90508	0,6	0,02	600000	508076	87653
3	3	7	0,90704	0,7	0,01	600000	558031	96859
4	1,5	8	0,91087	0,8	0,01	600000	431600	203704
5	2,5	4	0,91988	0,9	0,02	600000	439430	201656
6	3,5	5	0,92814	0,5	0,01	500000	334032	209639
7	1	6	0,93569	0,6	0,01	500000	332647	211770
8	2	7	0,94257	0,7	0,02	500500	335680	194984
9	3	8	0,94882	0,8	0,02	500000	462065	46550
10	1,5	4	0,95450	0,9	0,01	400000	364565	38410
11	2,5	5	0,95964	0,5	0,02	400000	373035	36802
12	3,5	6	0,96427	0,6	0,03	400000	372280	35494
13	1	7	0,96844	0,7	0,01	300000	257280	34688
14	2	8	0,97219	0,8	0,02	300000	258030	35078
15	3	4	0,97555	0,9	0,02	600000	557280	34688

Продолжение таблицы 7.2 – Индивидуальные исходные данные

Номер варианта	Средний интервал поступления запросов IntPostZ, мин	Ёмкость входного буфера, emkBuf	Доверительная вероятность α	Вероятность обработки запросов сервером ρ	Точность, вычислений ε	Производительность сервера Q, оп/сек	Математическое ожидание S1, оп	Среднеквадратическое отклонение S2, оп
16	1,5	5	0,97855	0,5	0,01	600000	508076	87653
17	2,5	6	0,98123	0,6	0,02	600000	558031	96859
18	3,5	7	0,98360	0,7	0,03	600000	431600	203704
19	1	8	0,98571	0,8	0,01	600000	439430	201656
20	2	4	0,98758	0,9	0,02	500000	334032	209639
21	3	5	0,98923	0,5	0,03	500000	332647	211770
22	1,5	6	0,99068	0,6	0,01	500500	335680	194984
23	2,5	7	0,99195	0,7	0,02	500000	462065	46550
24	3,5	8	0,99307	0,8	0,03	400000	364565	38410
25	1	4	0,93569	0,9	0,01	400000	373035	36802
26	2	5	0,94882	0,5	0,02	400000	372280	35494
27	3	6	0,95450	0,6	0,03	300000	257280	34688
28	1,5	7	0,96427	0,7	0,01	300000	258030	35078
29	2,5	8	0,96844	0,8	0,02	500500	335680	194984
30	3,5	4	0,97219	0,9	0,01	500000	462065	46550

Таблица 7.3 - Значения функции Лапласа

α	$t\alpha$	α	$t\alpha$	α	$t\alpha$
0,90106	1,65000	0,94882	1,95000	0,98123	2,35000
0,90508	1,67000	0,95450	2,00000	0,98360	2,40000
0,90704	1,68000	0,95964	2,05000	0,98571	2,45000
0,91087	1,70000	0,96427	2,10000	0,98758	2,50000
0,91988	1,75000	0,96844	2,15000	0,98923	2,55000
0,92814	1,80000	0,97219	2,20000	0,99068	2,60000
0,93569	1,85000	0,97555	2,25000	0,99195	2,65000
0,94257	1,90000	0,97855	2,30000	0,99307	2,70000

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Понятие имитационного моделирования
2. Что понимается под характеристикой эффективной работы СМО?
3. Случайные процессы какого типа протекают в СМО?
4. Что понимается под СМО с отказами?
5. Физический смысл приведенной интенсивности потока заявок
6. Расчет показателей эффективности одноканальной СМО с отказами.
7. Расчет показателей эффективности одноканальной СМО с ограниченной очередью.
8. Почему в среде AnyLogic в объекте Source вместо времени между прибытиями соседних заявок exponential (120) нужно указать exponential (1/120.0) ?
9. Какой объект предназначен для ввода запросов в модель в среде AnyLogic?
10. Какой объект предназначен для вывода запросов из модели в среде AnyLogic?
11. Какой объект имитирует очередь запросов в среде AnyLogic?
12. Какой объект имитирует обработку запросов в среде AnyLogic?