**Лабораторная работа № 5**

**Дискретно-событийное моделирование**

**Цель работы:**

1. Изучить основные понятия теории систем массового обслуживания.
2. Изучить возможности AnyLogic для моделирования систем массового обслуживания.
3. **Постановка задачи**

Сервер обрабатывает запросы, поступающие с автоматизированных рабочих мест с интервалами, распределенными по показательному закону со средним значением 2 мин. Время обработки сервером одного запроса распределено по закону со средним значением 3 мин. Сервер имеет входной буфер емкостью 5 запросов. Построить имитационную модель для определения математического ожидания времени обработки запросов и вероятности обработки запросов.

Сервер представляет собой одноканальную систему массового обслуживания (рис. 5.1) с ограниченной очередью (надежность системы будем считать абсолютной).



Рис. 5.1

1. **Построение модели**
2. Создайте модель с именем Server. Задайте единицы модельного времени – секунды.
3. Зададим динамику процесса, создав диаграмму из блоков **Библиотеки моделирования процессов** (рис. 5.2). Перетащите блоки на диаграмму (соединительные линии появляются во время перетаскивания) как показано на рис. 5.1.

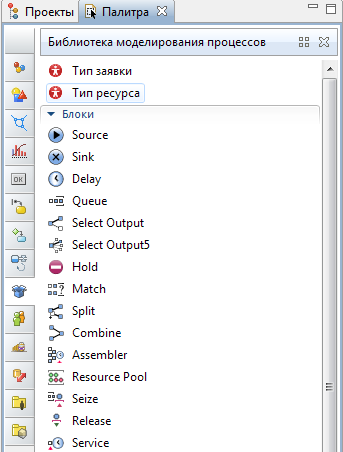


Рис. 5.2

Основные характеристики элементов созданной диаграммы:

**Source** генерирует заявки определенного типа. Обычно он используется в качестве начальной точки диаграммы процесса. В данном примере заявками будут запросы на обработку сервером, а объект **Source** будет моделировать их поступление. Имеет выходной **out** порт.



 **Queue** моделирует очередь заявок, ожидающих приема. В данном примере он будет моделировать очередь запросов, ожидающих освобождения сервера. Имеет четыре порта: входной порт **in**; выходной порт **out**; выходной порт для заявок, покидающих объект по таймауту (вследствие и стечения заданного времени ожидания) **outTimeout**; выходной порт для заявок, покидающих объект в результате вытеснения **outPreempted**.

 Объект **Delay** задерживает заявки на заданный период времени. Представляет в данной модели сервер, обрабатывающий запросы. Имеет два порта: входной порт **in**; выходной порт **out**.

 **Sink** уничтожает поступившие заявки. Обычно он используется в качестве конечной точки потока заявок (и диаграммы процесса соответственно). Имеет входной порт **in**.

http://127.0.0.1:49837/help/topic/com.xj.anylogic.help/html/interaction/images/PortWithQueue_obj.gif **Порты** играют центральную роль в механизме передачи сообщений. Сообщения посылаются и получаются портами. Порты являются двунаправленными и могут служить одновременно как для приема, так и для посылки сообщений. Обмен сообщениями возможен только между портами, соединенными **Соединителями**. Обратим внимание, что **Порт** и **Соединитель −** элементы  палитры  **Основная.**

1. Изменим свойства объектов:

* Выделите объект **source**. В таблице свойств в выпадающем списке **Прибывают согласно:** укажите, что запросы поступают согласно **Времени между прибытиями:** (рис. 5.3). В поле **Время между прибытиями** появится запись exponential(1). Установите согласно постановке задачи среднее значение интервалов времени   
  (2 мин) поступления запросов на сервер. Для этого вместо характеристики распределения 1 введите 1/120.0.
* Выделите объект **queue**. Задайте длину очереди. Введите в поле **Вместимость:** 5. В очереди будут находиться не более 5 запросов. Установите флажок **Включить сбор статистики**  (рис. 5.4). 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.

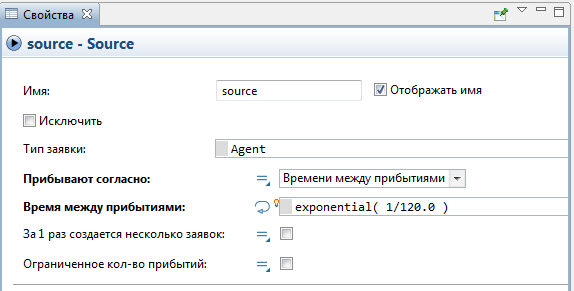


Рис. 5. 3

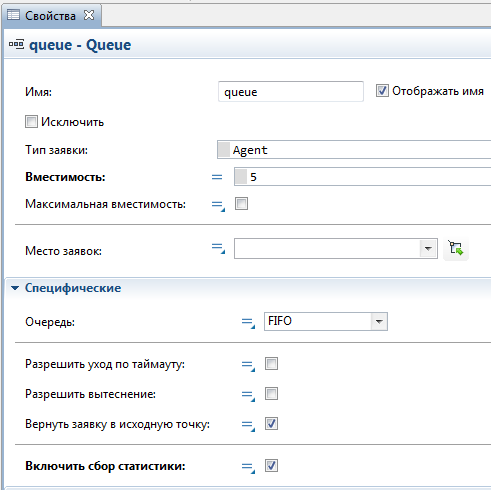


Рис. 5.4

* Выделите объект **delay**. Задайте время обслуживания, распределенное по экспоненциальному закону со средним значением 3 мин. Для этого введите в поле **Время задержки:** exponential(1/180.0). Установите флажок **Включить сбор статистики** (рис. 5.5). Для вывода коэффициента использования объекта **delay** в модели также следует предусмотреть соответствующий Java код.

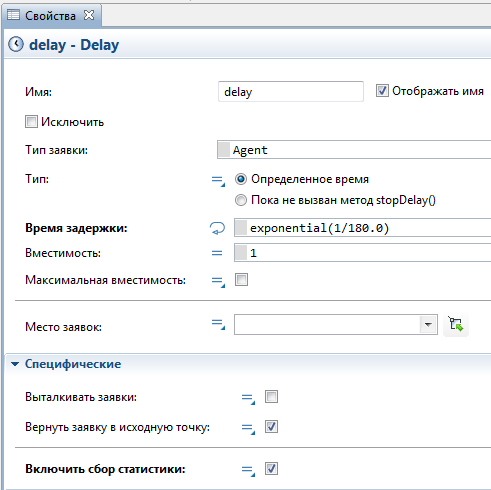


Рис. 5.5

* Объект **sink** выводит обработанные сервером запросы, его

оставим без изменения.

4. Установите свойства эксперимента. Для этого:

* В панели **Проект** выделите эксперимент **Simulation: Main**.
* Во вкладке **Модельное время** в поле **Остановить** выберите из списка **В заданное время.** В поле **Конечное время:** установите 3600 (обработку запросов сервером исследуем в течение одного часа, т.е. 3600 с.).
* Раскройте вкладку **Случайность**. Выберите опцию **Фикси-рованное начальное число (воспроизводимые прогоны)**. В поле **Начальное число:** установите 9.

5. Запустите модель на выполнение. Модель остановится с ошибкой (рис. 5.6), которая означает, что запрос не может покинуть объект source и войти в блок queue, так как его емкость, равная 5, заполнена (рис. 5.7).

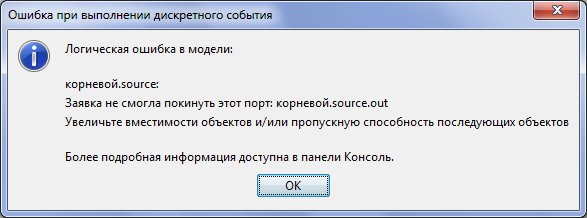


Рис. 5.6

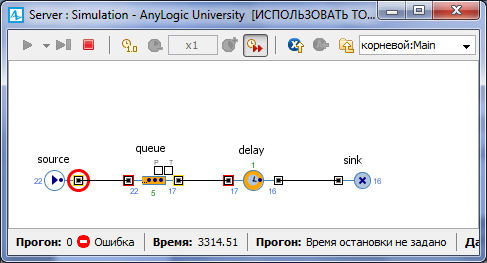


Рис. 5.7

Увеличьте длину очереди. Для этого введите в поле **Вместимость** 15. Снова запустите модель. Увеличьте скорость выполнения модели. Можно следить за состоянием любого объекта диаграммы процесса во время выполнения модели, с помощью окна инспекта этого объекта. В окне инспекта будет отображена базовая информация: 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.

Для предотвращения остановок модели из-за недостаточной емкости объекта queue, ёмкость его была увеличена и этим нарушено условие задачи. Выполнить его можно было бы изменяя среднее время поступления запросов и среднее время обработки запросов сервером, оставляя неизменной длину очереди. Как это сделать, будет рассмотрено позже.

1. **Создание анимации модели**

Визуализируем процесс поступления запросов на сервер и обработки запросов сервером. Так как в данном случае конкретное расположение объектов в пространстве не важно, то можно просто добавить схематическую анимацию моделируемых объектов − сервера и очереди запросов к нему.

1. Откройте палитру **Разметка пространства,**  содержащую в качестве элементов различные примитивные фигуры, используемые для рисования презентаций моделей - путь, прямоугольный узел, многоугольный узел, точечный узел, аттрактор, стеллаж, масштаб (рис. 5.8).

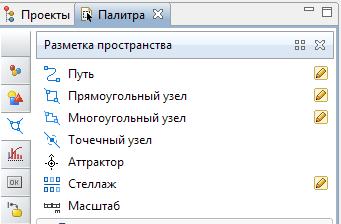


Рис. 5.8

1. Выделите элемент **Прямоугольный узел** и перетащите его на диаграмму класса активного объекта. Поместите элемент **Прямоугольный узел** так, как показано на рис. 5.9

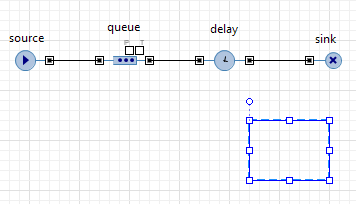


Рис.5.9

1. Сделаем так, чтобы цвет этого прямоугольного узла менялся в зависимости от того, обрабатывет ли сервер в данный момент времени запрос или нет. Выделите фигуру, перейдите на страницу **Внешний вид** панели свойств (рис. 5.10). В поле **Цвет заливки:** по стрелке, выберите **Динамическое значение** и введите код:

delay.size()>0?red:green.

Функция size() возвращает число запросов, обслуживаемых в данный момент времени. Если сервер занят, то цвет кружка будет красным, в противном случае − зелёным.

1. Нарисуйте путь, который будет обозначать на анимации очередь к серверу (рис. 5.11). Чтобы нарисовать путь, сделайте двойной щелчок мышью по элементу **Путь** палитры **Разметка пространства** (чтобы перейти в *режим рисования*). Теперь можно рисовать путь точка за точкой, последовательно щелкая мышью там, где будут располагаться вершины пути. Чтобы завершить рисование, добавьте последнюю точку пути двойным щелчком мыши. Очень важно, какая точка пути создается первой. Заявки будут располагаться вдоль нарисованного пути в направлении от конечной точки к начальной. Поэтому рисование пути обязательно начните слева.

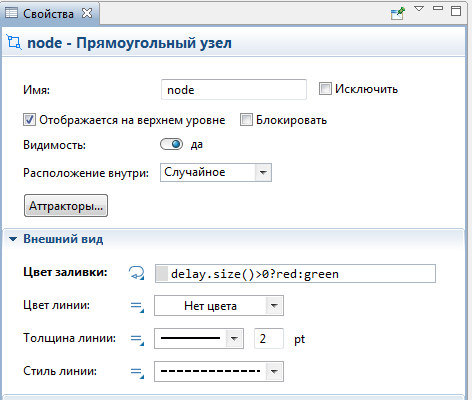


Рис. 5.10

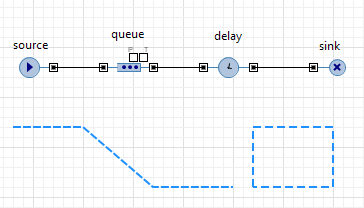


Рис. 5.11

1. Задайте путь в качестве фигуры анимации очереди. Выделите объект queue. На странице свойств объекта queue в поле **Место заявок:** выберите из выпадающего списка path (рис.5.12).

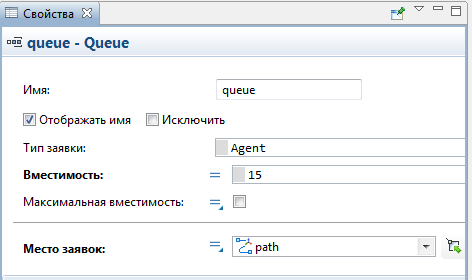


Рис. 5.12

1. Задайте прямоугольный узел в качестве фигуры анимации сервера. Выделите объект delay. Введите в поле **Место заявок:** из выпадающего списка имя прямоугольного узла: node (рис.5.13).

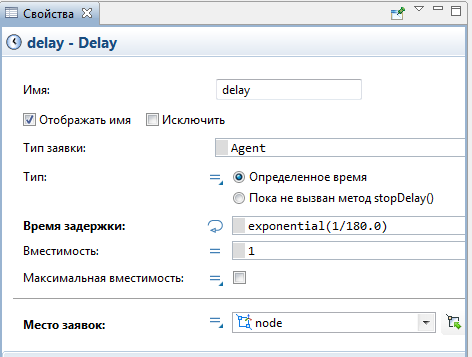


Рис. 5.13

1. Запустите модель. У модели теперь есть простейшая анимация − сервер и очередь запросов к нему (рис.5.14). Цвет фигуры сервера будет меняться в зависимости от того, обрабатывается ли запрос в данный момент времени или нет.

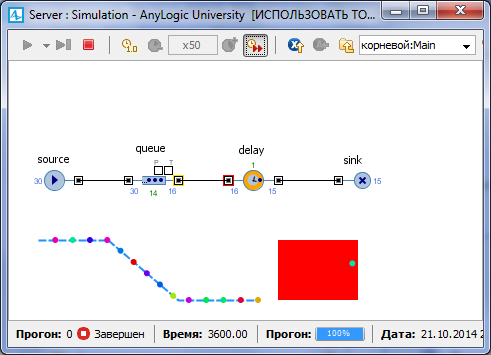


Рис. 5.14

1. **Сбор статистики использования ресурсов**
2. Создадим диаграмму, отображающую занятость сервера. Перетащите элемент **Столбиковая диаграмма** из палитры **Статистика** на диаграмму класса и измените ее размер, как показано на рис. 5.15.
3. Перейдите на панель **Свойства**. Щёлкните кнопку **Добавить элемент данных**. Измените **Заголовок** на SERVER. Введите delay.statsUtilization.mean() в поле **Значение** (рис.5.16). У объекта **delay** есть встроенный набор данных statsUtilization, занимающийся сбором статистики использования этого объекта. Функция mean() возвращает среднее из всех измеренных этим набором данных значений. Можно использовать и другие методы сбора статистики, такие, как min() или max().
4. Добавьте еще одну столбиковую диаграмму (рис. 5.17) для отображения средней длины очереди. Измените  **Заголовок:** наQueue lenqth и **Значение:** наqueue.statsSize.mean().

В секции **Внешний вид** установите горизонтальное размещение столбцов.

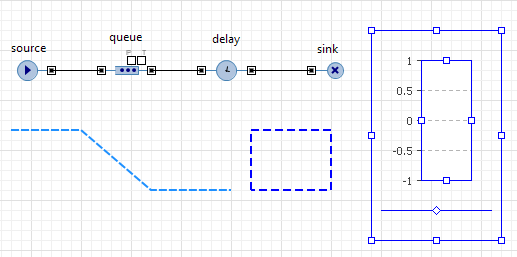


Рис. 5.15

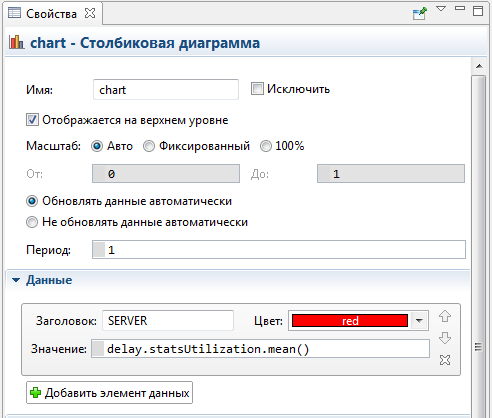


Рис.5.16

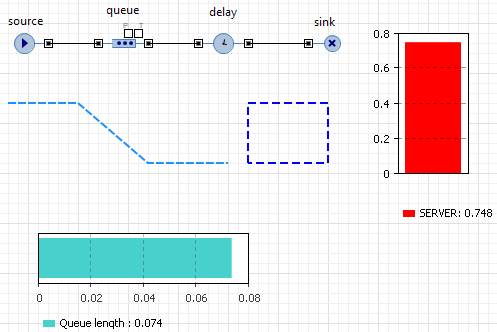


Рис. 5.17

4. Запустите модель с двумя столбиковыми диаграммами, установив модельное время 3600 единиц, и понаблюдайте за её работой (рис.5.18).

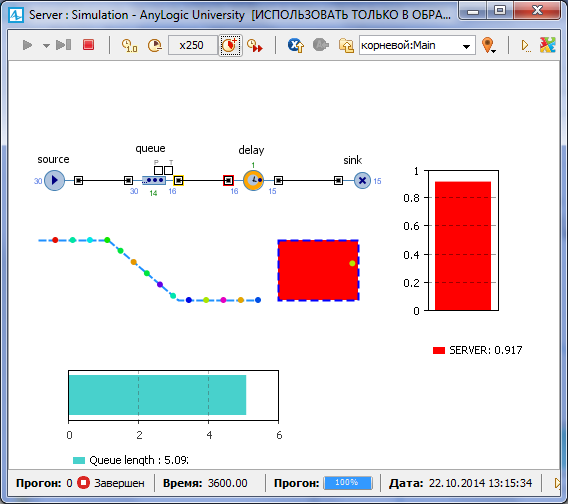


Рис. 5.18

**5.Уточнение модели согласно постановке задачи**

В постановке задачи емкость буфера была определена в 5 запросов. Но при такой емкости возникала ошибка — в какой-то момент времени очередной запрос не мог покинуть блок source, так как длина очереди была ограничена. Рассмотрим способы, позволяющие выполнить заданное условие не увеличивая емкость буфера.

Объект queue моделирует очередь заявок, ожидающих приёма объектами, следующими за ним в потоковой диаграмме, или же моделирует хранилище заявок общего назначения. При необходимости можно задать максимальное время ожидания заявки в очереди. Также можно с помощью дополнительно созданного программного кода извлекать заявки из любых позиций в очереди.

Заявка может покинуть объект queue различными способами:

* обычным способом через порт out, когда объект, следующий в диаграмме 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 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 будет вытеснена новая заявка*.

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

Уточните модель.

1. Выделите объект queue. На панели **Свойства** измените **Вместимость** с 15 на 5 запросов. Здесь же установите **Разрешить вытеснение**.

2. Для уничтожения потерянных запросов вследствие полного заполнения queue нужно добавить второй объект sink (рис. 5.19). Для этого сделайте двойной щелчок мышью по порту outPreempted, затем щёлкните в месте изгиба соединителя, закончите рисование соединителя двойным щелчком в месте предполагаемого соединения с портом in блока sink1, который нужно перетащить из **Палитры Библиотеки моделирования процессов** в завершении данной процедуры.

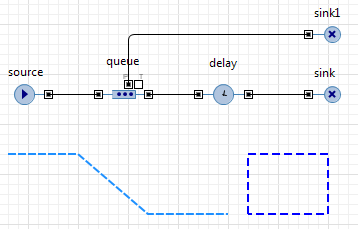


Рис. 5.19

Запустите уточненную модель и понаблюдайте за ее работой. Запросы при длине очереди равной пяти теряются, и ошибки при этом не возникает. Модель по ограничению ёмкости входного буфера и значениям других параметров соответствует постановке задачи.

**6. Сбор статистических данных об обработанных запросах**

Математическое ожидание или среднее время обработки одного запроса определяется как отношение суммарного времени обработки запросов к их количеству. Введем обозначения:

timeIn — время входа запроса в буфер сервера,

timeOut — время выхода запроса с сервера (входа в блок sink),

timeProcess — время обработки запроса,

тогда timeProcess=timeOut-timeIn.

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). Введем обозначения:

total— количество всех поступивших запросов,

served— количество обработанных сервером запросов,

probability — вероятность обработки запросов,

тогда probability=served/ total.

Entity являются базовым классом для всех заявок, которые создаются и работают с ресурсами в процессе, описанном с помощью диаграммы из объектов **Библиотеки моделирования процессов**. Entity 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 обработки запросов сервером и вероятности обработки запросов сервером. AnyLogic предоставляет возможность создавать запросы с необходимыми параметрами. Создадим нестандартный тип заявки Inquiry:

1. Откройте палитру **Библиотека моделирования процесов**.

2. Перетащите элемент **Тип заявки** в графический редактор   
(рис. 5.20). Появится диалоговое окно **Создание агента** (рис.5.21).

3. В поле **Имя нового агента:** введите Inquiry. Выберите анимацию агента: установите 2D и выберите из выпадающего списка элемент **Сообщение**. Щёлкните **Далее**.В окне **Параметры агента** добавьте следующие параметры: timeIn, total, served типа double.



Рис. 5.20

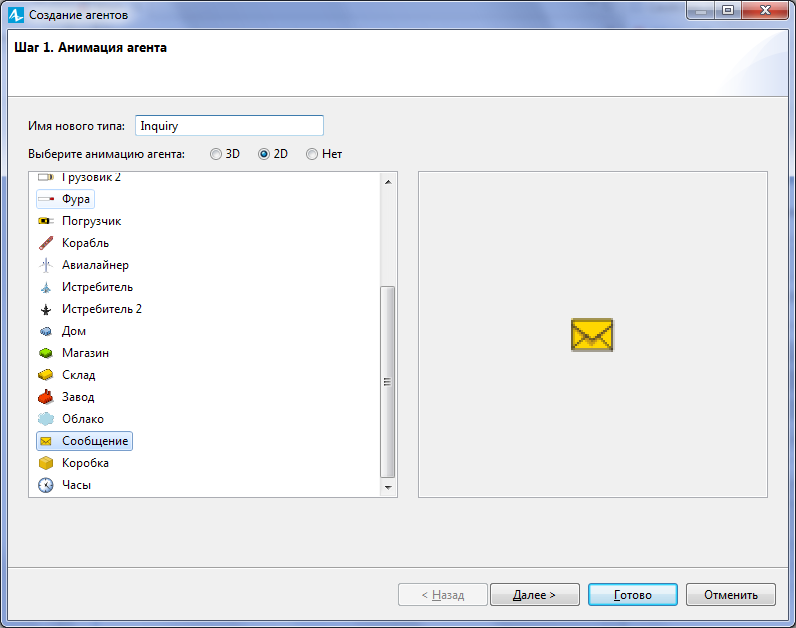


Рис. 5.21

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

Обратите внимание, что поле timeOut отсутствует, вместо него в дальнейшем будет использована функция time().

4. Щёлкните кнопку **Готово**. Вы увидите окно, в котором будут показаны автоматически созданные параметры нестандартного типа заявок Inquiry (рис. 5.22). Закройте окно, щелкнув крестик в закладке рядом с его названием.

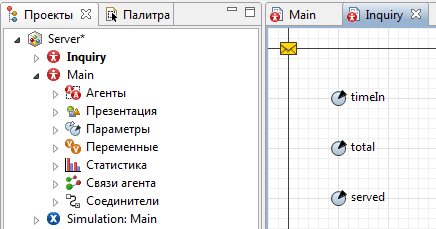


Рис. 5.22

**7. Изменение свойств объектов диаграммы**

Чтобы создавать заявки нестандартного типа Inquiry, нужно поместить вызов конструктора этого типа в поле **Новая заявка** объекта source. После этого действия заявки в потоке будут типа Inquiry, но объекты диаграммы будут продолжать их считать заявками типа Entity. Поэтому нужно указать имя нестандартного типа заявки в качестве **Типа заявки** у всех объектов потоковой диаграммы. С учётом блока sourceихвсего пять.

Измените их свойства.

1. Измените свойства объекта source (рис. 5.23): введите Inquiry в поле **Тип заявки:**. Это позволит напрямую обращаться к полям типа заявки Inquiry в коде динамических параметров этого объекта; выберите из выпадающего списка Inquiry() в поле **Новая заявка:**. Теперь этот объект будет создавать заявки типа Inquiry; введите entity.timeIn=time(); в поле **Действия при выходе:**. Код будет сохранять время создания заявки-запроса в параметре timeIn. Напомним, что функция time() возвращает текущее значение модельного времени.

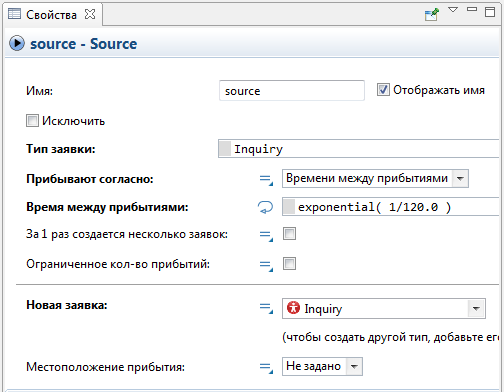


Рис. 5.23

2. Измените свойства объектов queue, delay, sink1, sink: введите Inquiry в поле **Тип заявки:**.

3. Организуем сбор данных о времени обработки запросов сервером. Элемент **Данные гистограммы** будет запоминать значения времен для каждого запроса. На основе этого он предоставит пользователю стандартную статистическую информацию (среднее, минимальное, максимальное из измеренных значений, среднеквадратичное отклонение и т.д.). Перетащите элемент **Данные гистограммы** с палитры **Статистика** на диаграмму активного класса (рис. 5.24). Задайте **Свойства** элемента: измените **Имя:** на timeProcess, **Кол-во интервалов:** 50, задайте **Нач. размер интервала:** 0.01.



Рис.5.24

4. Добавьте еще один элемент сбора статистики для определения вероятности обработки запросов: перетащите элемент **Данные гистограммы** на диаграмму активного класса. Задайте свойства элемента: измените **Имя:** на probability, **Кол-во интервалов:**  50, **Нач. размер интервала:** 0.01.

5. Теперь свяжем **Данные гистограммы**  timeProcess и probability с объектами процессной диаграммы: в объекте sink введите в поле **Действие при входе** следующие коды (рис. 5.25):

* timeProcess.add(time()-entity.timeIn);

Этот код добавит время обработки одного запроса в объект сбора данных гистограммы timeProcess. Данное время определяется как разность между текущим модельным временем time() и временем генерации запроса (add − функция добавления элемента в массив).

* entity. served =sink.count(); entity.total=source.count();

Эти код заносит количество запросов, вошедших в блок sink и вышедших из блока source соответственно (count() − функция, возвращающая количество значений, хранимых в объекте).

* probability.add(entity. served /entity.total);

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

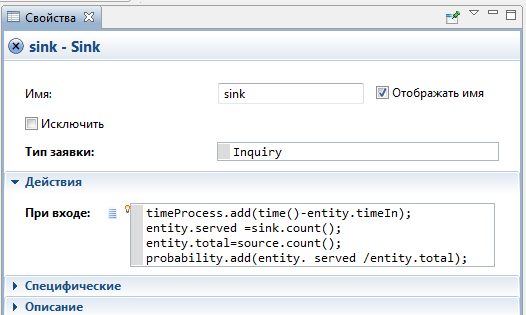


Рис.5.25

Итак, все условия постановки задачи выполнены. Чтобы наблюдать за работой модели, установите, что время остановки модели не задано. Запустите модель. Фрагмент работы показан на рис.5.26.

1. **Добавление гистограмм**

1. Перетащите элемент **Гистограмма** из палитры **Статистика** в поле графического редактора.

2. Укажите имя элемента сбора данных, которые необходимо отобразить на гистограмме: щёлкните кнопку **Добавить данные** и введите в поле **Данные** имя: timeProcess (рис.5.27). Установите **Отображать среднее**. В поле **Заголовок:** введите Histogram timeProcess.

3. Аналогично добавьте еще одну гистограмму. **Данные**: probability. Установите **Отображать среднее**. В поле **Заголовок:** введите Histogram probability.

4. Установите **Время остановки не задано**. Запустите модель. Фрагмент работы показан на рис.5.28.

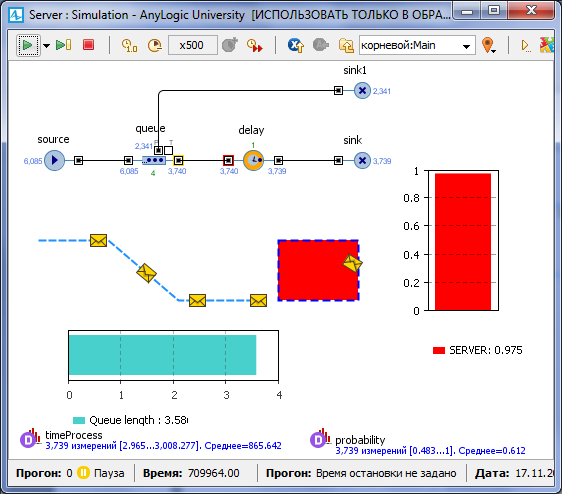


Рис. 5.26

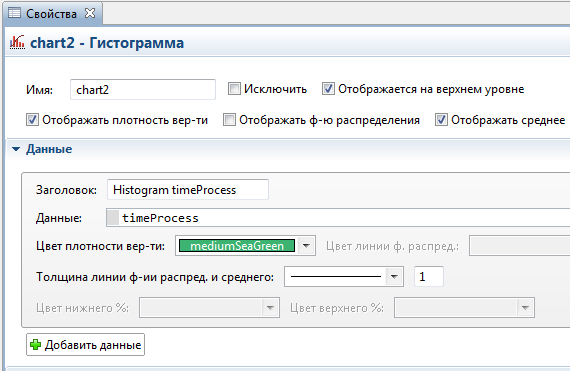


Рис. 5.27

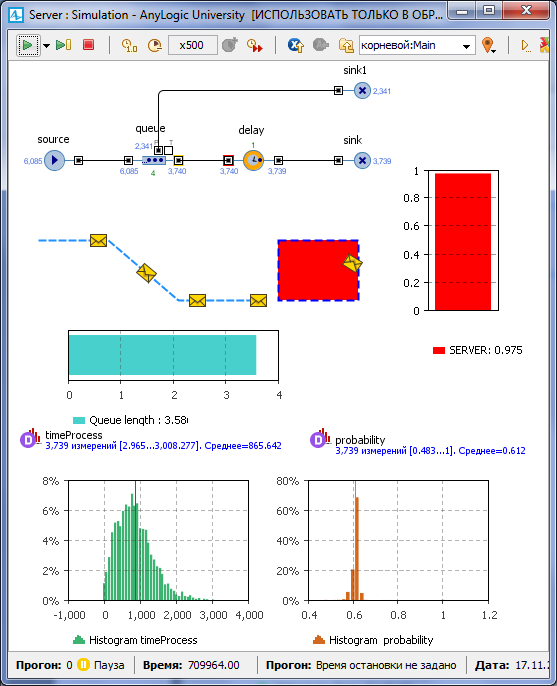


Рис. 5.28

**9. Изменение времени обработки запросов сервером**

В построенной модели, с целью упрощения процесса, время обработки запросов сервером было принято распределённым по показательному (экспоненциальному) закону со средним значением 3мин. Однако время обработки поступающих запросов зависит от производительности сервера Q=6·105оп/с и вычислительной сложности запросов, распределенной по нормальному закону, с математическим ожиданием M=6·107оп и среднеквадратическим отклонением *S*=2·105оп. Кроме того, в модели определяется среднее количество запросов, обработанных за время моделирования 3600 с.

Внесем в модель изменения.

1. Из палитры **Основная** перетащите три элемента **Параметр** на диаграмму класса Main.

2. В поле **Имя** каждого из элементов введите М, S и Q соответственно. Выберите **Тип** double.

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

4. Создайте переменную с именем nEntity (элемент **Переменная**,палитра **Основная)** (рис. 5.29).

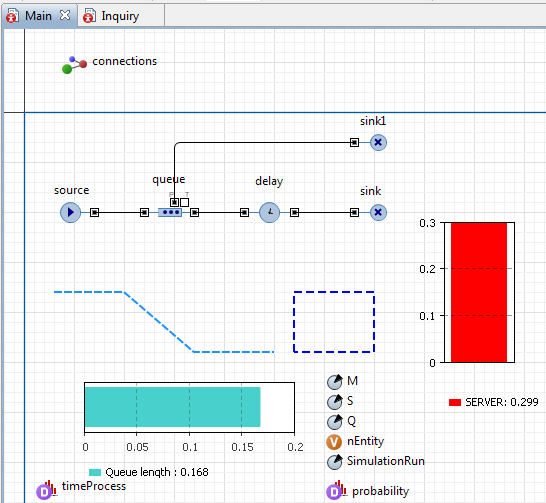


Рис. 5.29

5. Создайте параметр с именем SimulationRun. **Значение по умолчанию** задайте равным 9604.

Поясним назначение данного параметра.

Для получения результатов моделирования с доверительной вероятностью и точностью , нужно выполнить 9604 прогонов модели:



где − табулированный аргумент функции Лапласа, p − ожидаемая вероятность исхода события, в данном случае вероятность обработки запросов сервером.

Расчёт проведен для так называемого «худшего» случая, то есть в предположении, что ожидаемая вероятность обработки запросов *p*=0,5.

6. Выделите объект delay. В поле **Время задержки** вместо exponential (1/180.0) введите: (normal(S,M))/Q .

7. Выделите объект sink. В поле **Действие при входе** к имеющемуся там коду добавьте код:

nEntity=round(sink.in.count()/SimulationRun);

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.

8. В панели **Проект** выделите Simulation. На странице **Модельное время** в поле **Установить** выберите **В заданное время**. В поле **Конечное время** установите 34574400. Показатели моделируемой системы нужно определить в течение 3600 с, поэтому время моделирования составит 3600\*9604 = 34574400 единиц модельного времени.

9. Запустите модель. Результаты моделирования приведены   
на рис. 5.30.

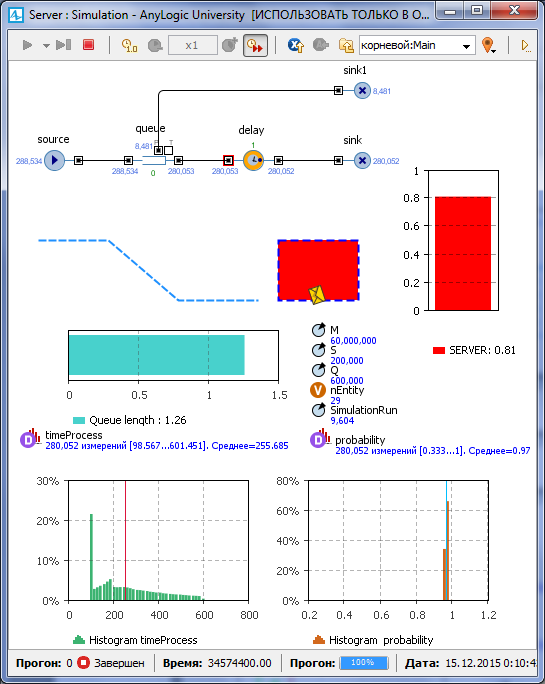


Рис. 5.30

**10. Задание для самостоятельной работы**

Проведите несколько экспериментов. Изменяйте среднее время поступления запросов tMean, емкость буфера emkBuf и производительность сервера Q. Результаты экспериментов внесите в таблицу 5.1. Сделайте выводы о том, как влияет изменение емкости входного буфера, интенсивности поступления запросов и производительность сервера на вероятность обработки запросов.

Таблица 5.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Параметры моделиро-вания | Коли-чество обрабо-танных запросов | Вероят-ность обработ-ки запросов | Среднее время обработ-ки одного запроса | Средняя длина очереди запросов  к серверу | Коэф-фициент исполь-зования сервера |
| tMean = 120  emkBuf = 5 |  |  |  |  |  |
| tMean = 120  emkBuf = 10 |  |  |  |  |  |
| tMean = 40  emkBuf = 5 |  |  |  |  |  |
| tMean = 40  emkBuf = 10 |  |  |  |  |  |
| tMean = 40  emkBuf = 10  Q=1000000 |  |  |  |  |  |
| tMean = 40  emkBuf = 15  Q=2000000 |  |  |  |  |  |

**Результаты работы**

Студент должен предоставить отчет по лабораторной работе с выводами, продемонстрировать работу модели, ответить на вопросы преподавателя.

**Контрольные вопросы**