Министерство образования и науки Российской Федерации

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. А.Н.ТУПОЛЕВА

Кафедра автоматики и управления

Математическое моделирование сложных систем

Лабораторный практикум
”

Составитель: Р.Н.Файзутдинов

Казань 2013

Учебное пособие ориентировано на изучение методов имитационного моделирования информационных, производственных и других сложных систем с помощью компонент **Simulink**, **SimEvents** и **Stateflow** программной системы Matlab.Содержит описание пяти лабораторных работ, посвященных общим принципам событийного моделирования в **Matlab 2013**.

Предназначено для магистров по специальности 220400.68 – Управление в технических системах при выполнении лабораторных работ по дисциплине «Математическое моделировании объектов и систем управления».

Лабораторная работа № 1.

Моделирование систем массового обслуживания в Simulink+ SimEvents

1. Системы массового обслуживания.

Для описания объектов реального мира, функционирующих в условиях действия случайных факторов на практике, часто используется класс математических моделей, называемых *системами массового обслуживания* (СМО). Функционирование этих объектов носит характер обслуживания поступающих в систему заявок или клиентов.

Для выполнения совокупности действий, включаемых в понятие “обслуживание”, система располагает наборами оборудования, называемых *обслуживающими каналами* или *линиями*. Например, в телефонных сетях заявками являются вызовы, возникающие на АТС в момент снятия абонентом телефонной трубки, а под обслуживанием понимается предоставление абоненту свободной линии для разговора; на бензозаправочной станции в качестве носителей заявок клиентов выступают автомобили, прибывающие на заправку, а обслуживающими каналами являются заправочные колонки. Аналогичные ситуации наблюдаются в системах посадки самолетов, разгрузки судов, в парикмахерских, магазинах и т.д. Любое производство можно представить в виде последовательности систем обслуживания.

Обычно предполагается, что заявки на обслуживание образуют *поток* – последовательность заявок со специальным чередованием моментов их появления во времени. Общее понятие потока основано на предположении, что интересующее нас событие будет происходить в заранее неизвестные моменты времени $t_1, t_2, \dots, t_k$ (обычно их считают случайными).

Математическое моделирование систем массового обслуживания складывается из моделирования потока заявок и моделирования процесса функционирования совокупности обслуживающих каналов.

Для описания СМО необходимо задать:

1) входящий поток требований или заявок, которые поступают на обслуживание;

2) порядок постановки заявки в очередь и выбора из нее;

3) правило, по которому осуществляется обслуживание;

Для описания *входящего потока* требований необходимо описать моменты времени их поступления в систему и количество требований, которое поступило одновременно. Закон поступления может быть детерминированный (например, одно требование поступает каждые 5 мин) или вероятностный (требование может появиться с равной вероятностью в интервале 5±2 мин). В общем случае входящий поток требований описывается распределением вероятностей интервалов времени между соседними требованиями.

Правила обслуживания характеризуются длительностью обслуживания (распределением времени обслуживания), количеством требований, которые поступили одновременно и дисциплиной обслуживания. Для характеристики свойств линии задается величина $\tau_з$ – длительность обслуживания заявки или *время занятости линии*, как случайная величина с заданным законом распределения.

В общем случае обслуживающая система может состоять из *n* линий, способных одновременно и независимо друг от друга обслуживать заявки. В любой момент времени линия находится в одном из двух состояний: свободном или занятом. Системы, обслуживаемые с помощью одной линии, называются *одноканальными системами*. Системы с несколькими линиями – *многоканальными*.

Относительно *порядка принятия* ***заявок*** в том случае, когда в системе имеется очередь заявок, используются следующие правила.

1. Заявки принимаются к обслуживанию в порядке очереди. Освободившаяся линия приступает к обслуживанию той заявки, которая ранее других поступила в систему. Такую дисциплину называют “раньше поступил – раньше обслужился” (в англоязычной литературе FIFO – First In - First Out).

2. Заявки принимаются к обслуживанию по минимальному времени получения отказа. Освободившаяся линия приступает к обслуживанию той заявки, которая в кратчайшее время может получить отказ.

3. Заявки принимаются к обслуживанию в случайном порядке в соответствии с заданными вероятностями (RANDOM).

Реальный процесс массового обслуживания состоит из последовательности фаз обслуживания, выполняемой различными устройствами. При этом следующее устройство приступает к обслуживанию заявки тогда, когда работа предыдущего с данной заявкой завершена.

Пример 1. Многофазное обслуживание покупателей.

1. Выбор товара и оформление товарного чека
2. Оплата в кассе.
3. Получение товара.

Пример 2. Технологический процесс.

Изделие может поступить на обработку станком (n+1)-й фазы лишь тогда, когда его обработка на станке n-й фазы закончена.

Система массового обслуживания – 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.1.1).

Имитационная модель СМО – это модель, отражающая поведение системы и изменения ее состояния во времени при заданных потоках заявок, поступающих на входы системы. Выходными параметрами являются величины, характеризующие качество функционирования системы, например, такие как

- коэффициенты использования каналов обслуживания;
- максимальная и средняя длина очередей в системе;
- время нахождения заявок в очередях и каналах обслуживания.

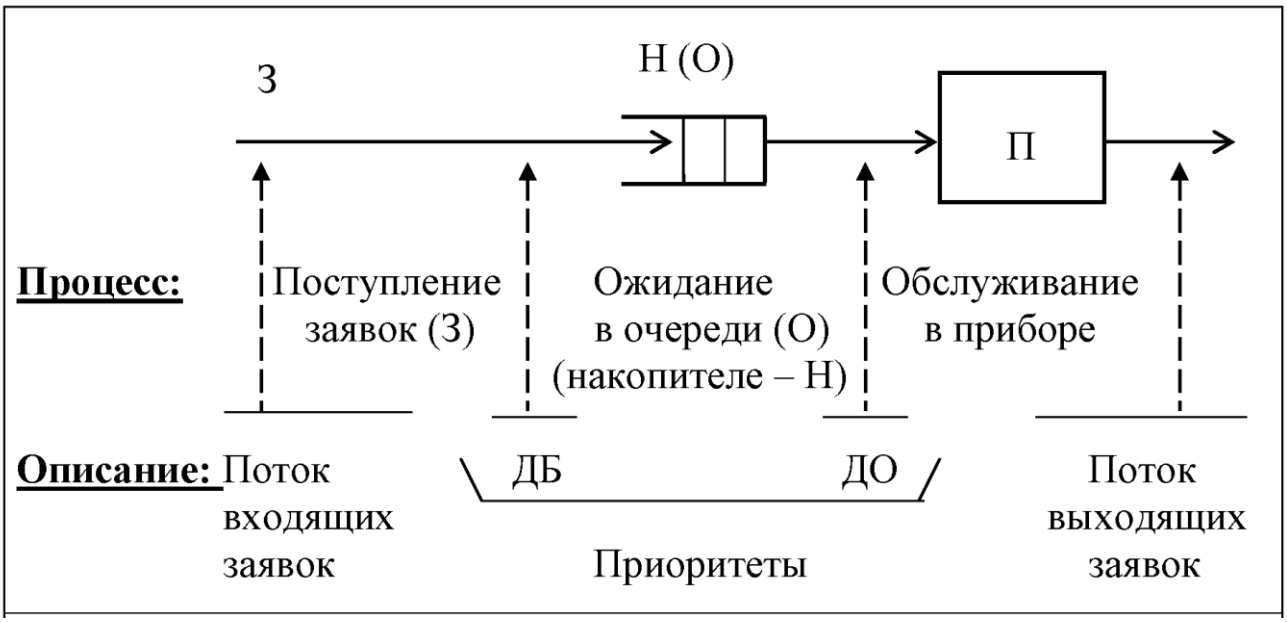


Рис. 1.1. Система массового обслуживания

2. Система дискретного событийного моделирования SimEvents.

Для реализации дискретно-событийного моделирования в среде **Simulink** используется компонента **SimEvents**. С помощью **SimEvents** можно моделировать и проектировать распределенные системы управления, аппаратные конфигурации, сети передачи и сбора информации для аэрокосмических, автомобильных, электронных и других приложений. Можно также моделировать событийно-управляемые процессы, такие например, как стадии производственного процесса для определения потребностей в ресурсах и оценки узких мест производства.

SimEvents и **Simulink** создают интегрированную среду для моделирования гибридных динамических систем, содержащих непрерывные компоненты и компоненты с дискретными событиями и дискретным временем.

Что такое моделирование дискретных событий? При моделировании дискретных событий или моделировании на основе событий состояния системы изменяются в результате наступления асинхронных дискретных инцидентов, которые называются событиями. В противоположность этому моделирование, основанное исключительно на дифференциальных или разностных уравнениях, в которых время является независимой переменной, – моделирование на основе времени, потому что состояния системы зависят от времени. **Simulink** предназначен для моделирования на основе времени, в то время как **SimEvents** создавался для моделирования дискретных событий. Выбор типа моделирования зависит как от самого изучаемого явления, так и от способа, которым его предполагается изучать.

При дискретно-событийном моделировании используется понятие *сущности* (*entity*). Сущности могут перемешаться через сети очередей (*queues*), серверов (*servers*) и переключателей (*switches*), управляемых дискретными событиями, в процессе моделирования.

Графические блоки **SimEvents** могут представлять компонент, который обрабатывает сущности, но сами сущности не имеют графического представления. Примеры сущностей приведены в следующей таблице.

Таблица 1. Примеры сущностей.

Приложение	Сущность
Аэропорт с очередью на взлетной полосе	Самолет, ждущий доступа к взлетной полосе
Вычислительная сеть	Пакеты, фреймы и сообщения для передачи
Конвейер для сборки агрегатов	Сборочные единицы

Событие (*event*) – это мгновенное дискретное явление, которое изменяет переменную состояния, выход и/или является причиной появления других событий. Примерами событий в модели **SimEvents** являются:

- перемещение сущности от одного блока к другому;
- завершение обслуживания сущности в сервере.

Для открытия библиотек SimEvents наберите в командной строке `simeventslib`.

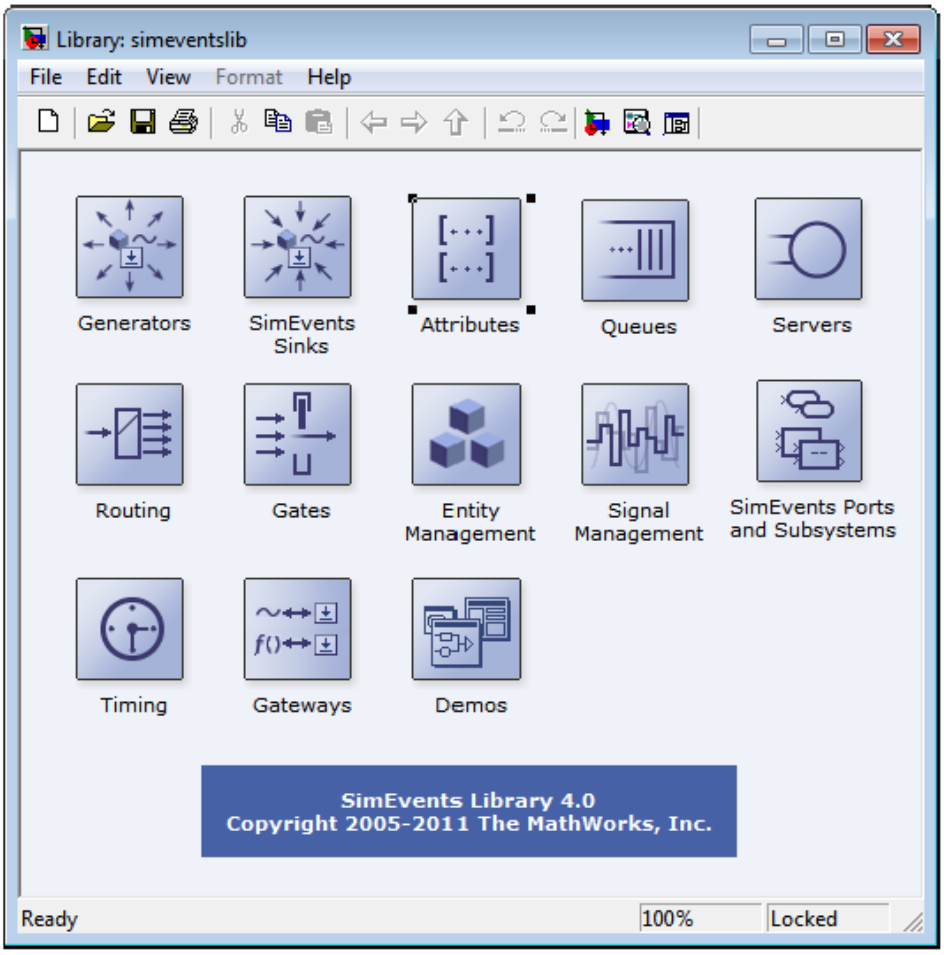


Рис.1.2. Окно библиотеки SimEvents.

В появившемся окне отображены иконки, представляющие все разделы библиотеки блоков `simevents`.

Выберем из раздела `Generators` (подраздел `Entity Generators` – генераторы сущностей) блок `Time-Based Entity Generator`, из раздела `Queues` блок `FIFO Queue`, из раздела `Servers` – блок `Single Server`, из раздела `SimEvents Sinks` – блок `Entity Sink` и блок `Signal Scope`. В результате в окне модели будет отображаться следующая картина.

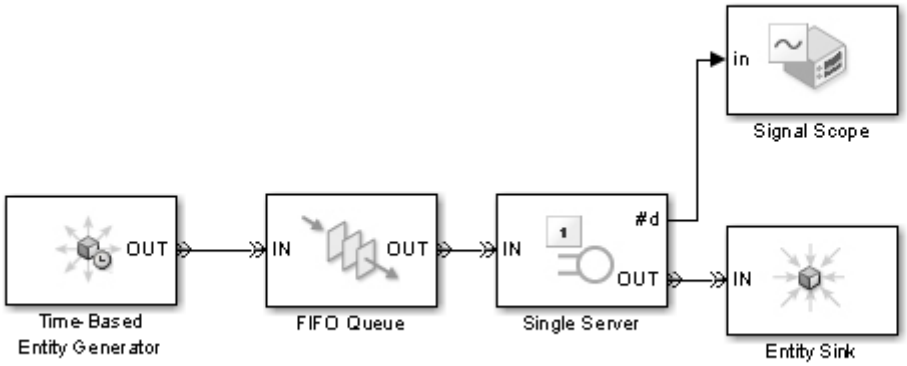


Рис. 1.3.

Представленная на рисунке модель содержит блоки для всех ключевых процессов моделирования: генерации сущностей (блок `Time-Based Entity Generator`), хранения сущностей в очереди (блок – `FIFO Queue`), обслуживания сущностей (блок – `Single Server`) и отображения информации о ходе моделирования (блок – `Signal Scope`).

При дискретно-событийном моделировании очереди (*queues*) хранят сущности в течение некоторого интервала времени, который заранее неизвестен. Очередь пытается выпустить сущность как можно быстрее, но успех операции зависит от возможности следующего блока принять новую сущность. Повседневным примером очереди является ситуация, когда вы стоите в очереди в кассу в магазине для обслуживания и не можете заранее определить сколько придется ждать обслуживания.

Отличительными свойствами очереди являются:

- емкость (*capacity*), то есть максимальное количество сущностей, котрые очередь может хранить одновременно;
- дисциплина очереди, определяющая какая из сущностей покинет очередь первой, если она хранит несколько сущностей.

Блоки очередей находятся в разделе `Queues` библиотеки `SimEvents`.

Сервер (*server*) в дискретно событийном моделировании хранит сущности в течение некоторого промежутка времени, называемого временем

обслуживания (*service time*) и затем пытается выпустить сущность. Во время периода обслуживания говорят, что блок-сервер обслуживает сущность.

Время обслуживания для каждой сущности вычисляется в момент ее прибытия в сервер. В отличие от этого время хранения блока в очереди принципиально заранее никогда неизвестно. Однако, если следующий блок не принимает сущность, которую уже обслужил сервер, то сервер должен хранить сущность дальше.

Отличительными свойствами сервера являются:

- число сущностей, которые можно обслужить одновременно. Это число может быть конечным или бесконечным;
- характеристиками или методами вычисления времен обслуживания поступающих сущностей;
- разрешает ли сервер прибывающим сущностям занимать сервер при наличии других сущностей, хранящихся в сервере. При отсутствии этого свойства сервер с конечной емкостью, если он полон, не принимает к обработке новые прибывающие сущности.

Блоки серверов находятся в разделе `Servers` библиотеки `SimEvents`.

3. Пример.

Требуется промоделировать работу парикмахерской. Интервалы прихода клиентов в парикмахерскую с одним креслом распределены равномерно на интервале 18±6мин. Время стрижки также распределено равномерно на интервале 16±4мин. Клиенты приходят в парикмахерскую, стригутся в порядке очереди: «первым пришел – первым обслужился». Промоделируйте работу парикмахерской в течение 8 часов.

Требуется определить параметры функционирования парикмахерской:

- коэффициент загрузки парикмахера;
- максимальное, среднее и текущее число посетителей в очереди;
- среднее время обслуживания.

Построение модели. Порядок блоков в модели соответствует порядку фаз, в которых клиент оказывается при движении в реальной системе:

1) клиент приходит;

2) если необходимо, ждет своей очереди;

3) садится в кресло парикмахера;

4) парикмахер обслуживает клиента;

5) клиент уходит из парикмахерской.

Модель представлена на рисунке 1.4.

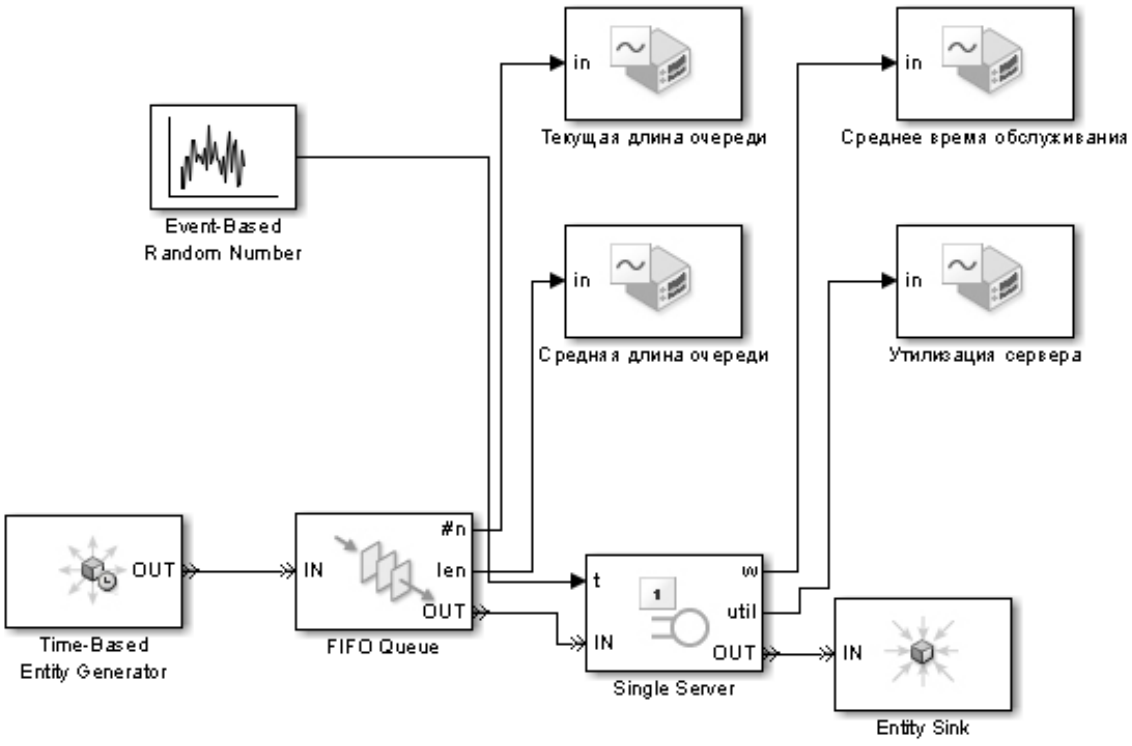


Рис. 1.4. Модель работы парикмахерсой.

В случайные моменты времени блок `Time-Based Entity Generator` генерирует события, моделирующие приход клиентов в парикмахерскую. В параметрах блока указан тип распределения `Uniform` (равномерный), параметры: **Minimum** – 12, **Maximum** 24.

Блок FIFO Queue сохраняет заявки (клиентов), которые парикмахер не может немедленно выполнить. Емкость блока принята бесконечной (inf).

Блок Single Server моделирует обслуживание клиента парикмахером. Этот блок может выполнить не более одной работы одновременно, тем самым ограничивая обработку новых работ. Времена обслуживания задаются через сигнальный порт t (**Service time from** – Signal port t) блоком Event Based Random Number, генерирующим равномерно распределенные случайные числа – тип распределения Uniform (равномерный), параметры: **Minimum** – 12, **Maximum** 20.

Линии для передачи сущностей (entity connection line) показывают связь между двумя блоками (или между их входными/выходными портами для сущностей) путем отображения пути, по которому сущность может:

- покинуть один из блоков;
- одновременно прибыть в следующий блок.

При моделировании сущность, которая покидает порт OUT одновременно прибывает в порт IN следующего связанного блока.

Линии для связи сущностей нельзя разветвлять. Если в приложении требуется, чтобы сущность прибыла в несколько блоков, для создания копий сущностей используется блок Replicate.

Блок Entity Sink поглощает работы, обработка которых завершена.

Часть блоков в рассматриваемой модели могут обрабатывать сигналы. Сигналы представляют собой численные величины, определенные в любой момент времени в течение процесса моделирования (не только в дискретные моменты времени). Сигналы появляются на соединительных линиях между сигнальными портами двух блоков. Например, сигнальный выходной порт (signal output port) #n блока FIFO Queue вязан с сигнальным входным портом (signal input port) in блока Текущая длина очереди (Signal Scope).

При моделировании событийно-управляемых систем, не содержащих блоков с непрерывными состояниями, необходимо настроить соответствующим образом параметры моделирования. Выберите команду `Simulation->Configuration parameters->Solver`. В разделе `Solver options` в поле `Type` выберите `Variable-step` и в поле `Solver – Discrete`. В поле `Max step size` (максимальный размер шага) введите `inf` (бесконечность).

Время моделирования принято 480 минут (8 часов).

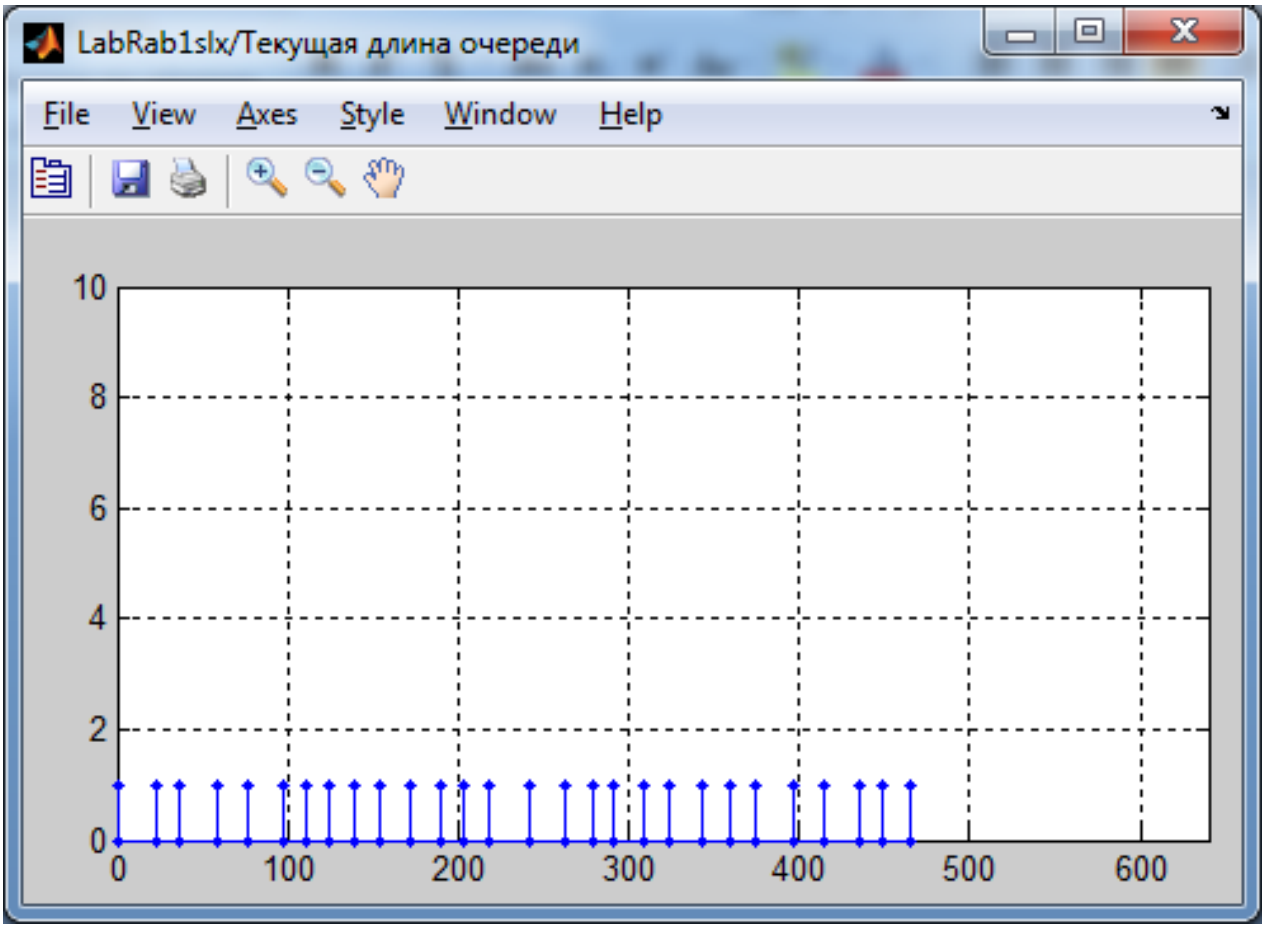


Рис. 1.5. Изменение длины очереди по времени.

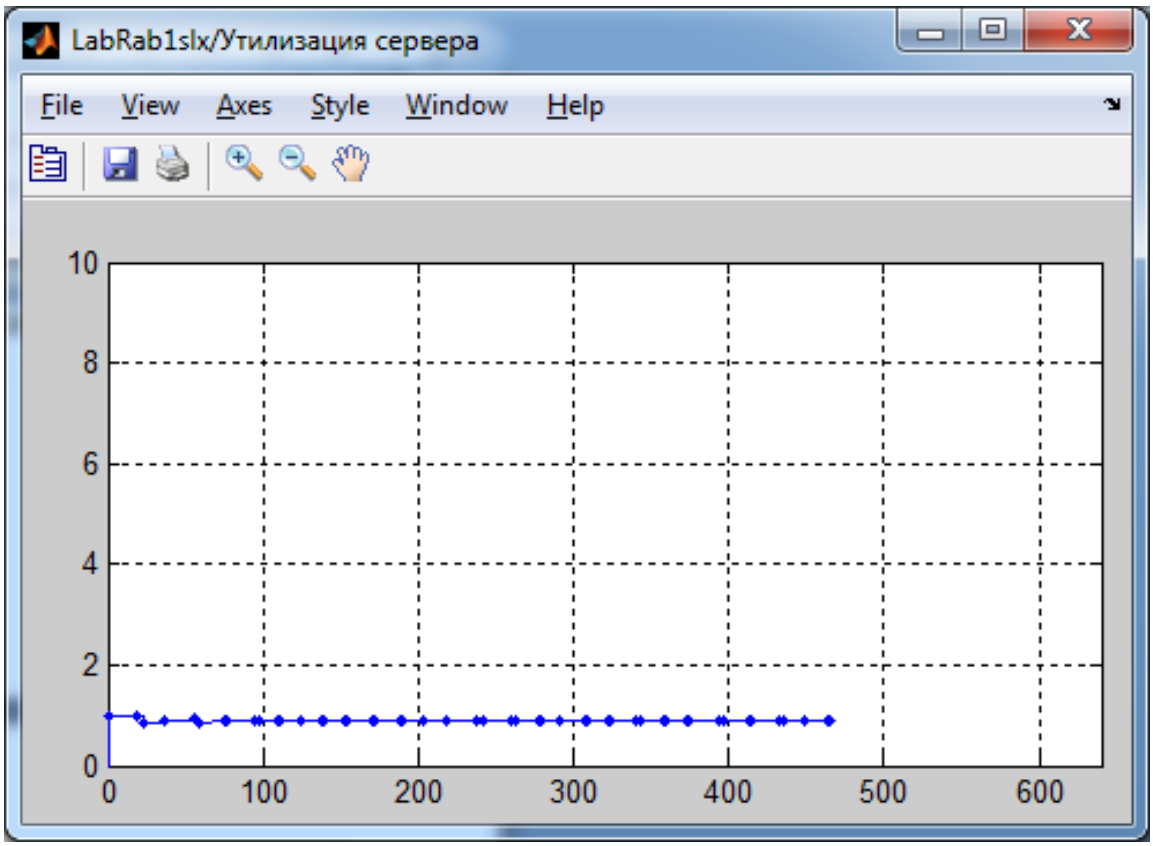


Рис. 1.6. Изменение загрузки парикмахера по времени

4. Описание используемых блоков.

Time-Based Entity Generator (Раздел Generators/Entity Generator.) – блок генерирует сущности в моменты времени, определяемые входным сигналом или статистическим распределением.

Порядок генерирования определяется значением параметра **Generate entities upon**:

Intergeneration time from dialog – моменты времени генерации определяются в зависимости от параметров в диалоговых полях блока;

Тип распределения моментов генерации определяется полем **Distribution**. Возможные значения:

- Constant (постоянное) – постоянное время между генерируемыми событиями задается в **поле Period.**
- Uniform (равномерное) – случайное равномерное распределение задается диапазоном в полях **Minimum** и **Maximum**.
- Exponential – экспоненциальное распределение задается параметром **Mean** (среднее).

При установке параметра распределения в значения Uniform или Exponential в диалоговом окне имеется также параметр **Initial seed,** определяющий генерируемый набор случайных чисел. Для фиксированного значения этого параметра случайная последовательность при следующем запуске модели повторится. Типично, значение этого параметра устанавливается в большое (например, пятизначное) нечетное число.

Intergeneration time from port t – моменты времени генерации определяются через сигнальный порт t, информация с которого считывается при старте моделирования и в каждый момент генерации новой

сущности. Сигнал должен быть событийным (event-based). Если выставлено это значение, то у блока появляется дополнительный сигнальный порт `t`.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке **Statistics** в панели свойств блока. Если отмечен соответствующий пункт, то у блока появляется новый сигнальный выходной порт, на который выводится следующая статистическая информация в соответствии:

`#d` – число сущностей покинувших блок после начала моделирования;

`w` – среднее время между генерациями сущностей.

`FIFO Queue` – блок одновременно хранит до `N` сущностей, где `N` – значение параметра **Capacity** (Емкость). Блок пытается выпустить сущность через выходной порт `OUT`, однако если порт `OUT` блокирован, то сущность остается в блоке. Если в блоке хранятся несколько сущностей, то сущности покидают блок в соответствии с дисциплиной первый вошел – первый вышел (first in – first out (FIFO)). Если блок уже хранит `N` сущностей, то входной порт `IN` блока не доступен.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке **Statistics** в панели свойств блока. Если отмечен соответствующий пункт, то у блока появляется новый сигнальный выходной порт, на который выводится следующая статистическая информация:

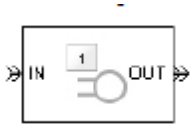
`#d` – число сущностей покинувших блок через порт OUT после начала моделирования;

`#n` – число сущностей в очереди;

`w` – среднее время ожидания в этом блоке для всех сущностей, покинувших блок через любой порт;

`len` – среднее число сущностей в очереди по времени, то есть средний по времени сигнал #n.

`Single Server` – блок обслуживает одновременно одну сущность за некоторый интервал времени и затем пытается выпустить сущность через выходной порт `OUT`. Если порт `OUT` блокирован, то сущность остается в блоке до тех пор пока выходной порт не разблокируется.



Время обслуживания (`Service Time`) определяется через параметры, атрибуты или сигналы в зависимости от значения параметра **Service Time From**. Блок определяет время обслуживания сущности при ее поступлении. Времена обслуживания определяются в секундах.

Значения параметра **Service Time From**:

`Dialog` – значения времени обслуживания задаются в поле параметра **Service Time**;

`Attribute` – значения времени обслуживания задаются атрибутом, имя которого указано в поле параметра **Attribute name**;

`Signal port t` – значения времени обслуживания задаются через сигнальный порт `t`. Сигнал должен быть событийным (event-based). Если выставлено это значение, то у блока появляется дополнительный сигнальный порт `t`.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке **Statistics** в панели свойств блока. Если отмечен соответствующий пункт, то у блока появляется новый сигнальный выходной порт, на который выводится следующая статистическая информация:

`#d` – число сущностей покинувших блок через порт OUT после начала моделирования;

`#n` – число сущностей в сервере, 0 или 1;

`w` – среднее время ожидания в этом блоке для всех сущностей, покинувших блок;

`util` – утилизация сервера, то есть доля времени моделирования, использованная на хранение сущностей. В начале моделирования утилизация равна 0 или 1 в зависимости от того хранит ли сервер сущность.

Для завершения путей сущностей используется блок `Entity Sink`.

Параметры блока:

Input port available for entity arrivals – если выбран этот параметр, то блок принимает прибывающие сущности, в противном случае – блок сущности не принимает, а при попытке прибытия сущности выдается сообщение об ошибке.

Report number of entities arrived, #a – при выборе параметра у блока появляется сигнальный порт `#a`, на котором после каждого прибытия сущности выдается информация о количестве принятых блоком сущностей. Начальное значение сигнала после начала моделирования до первого обновления блока равно 0.

Для вывода результатов моделирования в виде графиков используется блок `Signal Scope` из раздела библиотеки `SimEvents Sinks`.

Блок создает график, используя данные из событийного сигнала. Данные, откладываемые по вертикальной оси, берутся из сигнала, связанного с входным сигнальным портом `in`.

Параметры блока:

Plot type – тип графика:

`Stair` – ступенчатый;

`Continuous` – непрерывный. Тип графика не меняет дискретный сигнал на непрерывной, а только определяет способ отображения графика.

Данные для горизонтальной оси определяются параметром **X value from**: `Event time` – выводится график данных с порта `in` по времени моделирования; `Index` – выводится график данных с порта `in` по их индексу, т.е. первое значение имеет индекс 1, второе – 2 и т.д.

5. Задание. Моделирование работы магазина.

Требуется промоделировать работу небольшого магазина, который имеет один кассовый аппарат и одного продавца. Известны следующие параметры функционирования магазина:

- поток покупателей (заявок), приходящих в магазин за покупками, равномерный;
- интервал времени прибытия покупателей колеблется в пределах от 8,7 минуты до 10,3 мин. включительно, или 9,5 ± 0,8 мин;
- время пребывания покупателей у кассового аппарата составляет 2,3 ± 0,7 мин. После этого покупатели подходят к продавцу для получения товара;
- время, потраченное на обслуживание покупателей продавцом, составляет 10 ± 1,4 мин.

Требуется определить параметры функционирования магазина:

- коэффициент загрузки кассира;
- коэффициент загрузки продавца;
- максимальное, среднее и текущее число покупателей в каждой очереди;
- среднее время обслуживания в каждом канале обслуживания.

Лабораторная работа № 2.

Моделирование сборочного участка цеха в Simulink+ SimEvents

1. Постановка задачи.

На сборочный участок цеха предприятия поступают партии, каждая из которых состоит из трех деталей. Партии поступают через интервалы времени τ.

Половина всех поступающих деталей перед сборкой должна пройти предварительную обработку в течении времени τ_1. Предварительная обработка следующей детали осуществляется после завершения обработки предыдущей детали.

На сборку одновременно подаются обработанная и не обработанная детали. Процесс сборки занимает τ_2 единиц времени. В результате сборки получается изделие. Следующее изделие собирается после завершения сборки предыдущего изделия. В результате сборки возможно появление 30% бракованных изделий. Бракованные изделия исключаются из дальнейшего процесса.

Изделия без брака отправляются на регулировку, продолжающуюся в течении τ_3 единиц времени. Следующее изделие регулируется после завершения регулировки предыдущего изделия. Изделие, прошедшее регулировку, считается готовой продукцией.

Случайные величины τ, τ_1, τ_2, τ_3 имеют следующее распределение: Равномерное $0 \le x \le 5$.

Требуется моделировать работу участка на интервале времени $0 \le t \le$ 480 мин.

В результате моделирования требуется определить следующие характеристики сборки:

1. Количество заявок, завершивших обслуживание.
2. Количество заявок, получивших отказ.
3. Длины очередей в системе.

Рассматриваемая система реализуется на основе систем массового обслуживания следующим образом:

1. Заявкой является пара деталей, которые обрабатываются, собираются в изделие, которое либо попадает в брак, либо регулируется.
2. Входной поток заявок в систему обслуживания генерируется на основе исходного потока партий деталей. Далее каждая партия разделяется на три детали.
3. Сборочный участок цеха представляет собой одноканальную трехфазную СМО. Первая фаза - предварительная обработка. Вторая фаза – сборка. Третья фаза – регулировка.
4. Кроме одноканальных СМО, в формальную систему входят блоки формирования внутренних потоков заявок и логические блоки.

Процесс обслуживания заявок подчиняется следующим правилам:

1. В каждом из участков в данный момент времени может обслуживаться только одна заявка.
2. Заявки обслуживаются в порядке поступления в очередь.
3. Заявка, попавшая в брак, получает отказ.

Принципиальная схема системы сборочного производства имеет следующий вид:

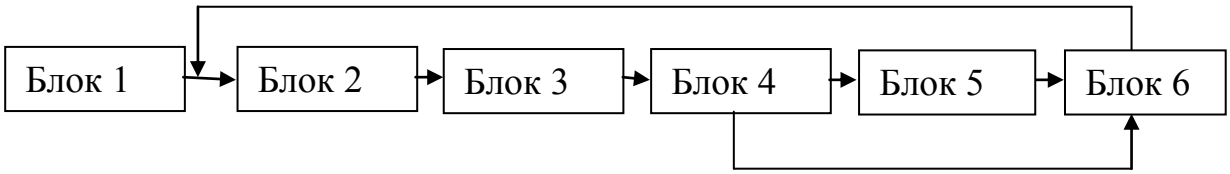


Рисунок 2.1 – Формализованная схема имитационной модели

Блок 1 – формирование входного массива пар деталей на основе исходного массива моментов времени поступления партий деталей.

Блок 2 – предварительная обработка заявки.

Блок 3 – сборка изделия.

Блок 4 – логический блок определения качества изделия.

Блок 5 – регулировка изделия.

Блок 6 – логический блок, определяющий переход к рассмотрению очередной заявки или завершение процесса анализа обработки заявок.

2. Определение путей для сущностей.

Путь для сущностей – это связь между выходным портом и входным портом для сущностей, определяющая эквивалентность между выходом сущности из одного блока и ее прибытием во второй блок.

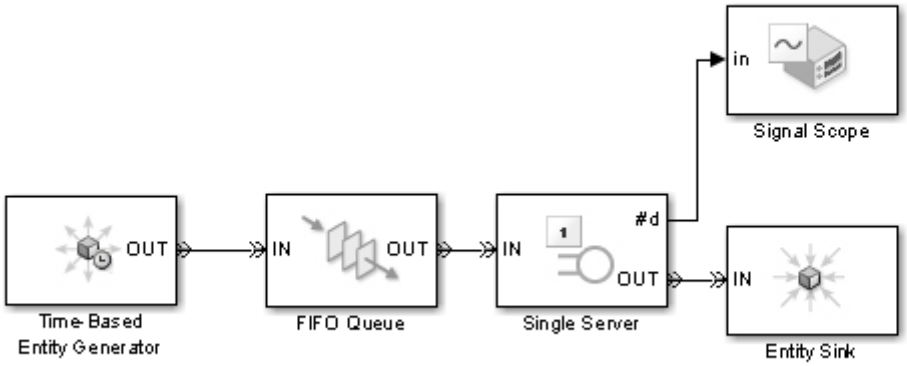


Рис. 2.2

Например, на рисунке выше любая сущность, покидающая выходной (OUT) порт блока `FIFO Queue`, эквивалентно прибывает во входной (IN) порт блока `Single Server`. Существование пути не гарантирует, что какая-либо сущность действительно использует этот путь. Например, время моделирования может быть настолько коротким, что никакая из сущностей не будет вообще сгенерирована. Даже если путь используется, то он используется только в дискретные моменты времени во время моделирования.

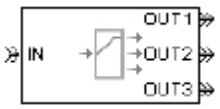
Проектирование путей для сущностей производится путем с использованием блоков из раздела `Routing` библиотеки `SimEvents`. Эти блоки имеют дополнительные порты для сущностей, которые позволяют менять топологию модели (то есть набор блоков и соединительных линий).

Типичными причинами, когда требуется манипулирование путями:

- Для описания параллельного поведения блоков при моделировании, например, для описания компьютерного кластера из двух компьютеров, разделяющих вычислительную нагрузку. В этой ситуации можно использовать блок `Output Switch` для посылки вычислительных работ на один из двух компьютеров. Можно также использовать блоки `Path Combiner` или `Input Switch`, если вычислительные работы после прохождения пары компьютеров следуют в одно место.
- Для описания нелинейных топологий, таких как обратная связь, например, при повторе операций, если критерий качества (например, качество обслуживания) не соблюдается. В этом случае можно использовать блок `Path Combiner` для объединения путей новых сущностей и сущностей, требующих повторного обслуживания.
- Для внедрения логических решения в процесс моделирования, например, для определения сценариев диспетчирования. В этом

случае можно использовать блок Input Switch для определения одной из нескольких очередей после прохождения сервера.

Блок Output Switch позволяет выбрать в процессе моделирования один из нескольких выходных портов для сущностей. После выбора одного из выходных портов, остальные порты становятся недоступными.



Количество выходных портов определяется параметром **Number of entity output ports**, входной порт, которые выбирается для приема сущностей определяется в соответствии с параметром **Switching Criteria**:

Round robin – в начале моделирования выбирается порт OUT1. При каждом следующем прибытии сущности выбирается следующий по порядку входной порт. После выбора последнего порта снова выбирается порт OUT1.

Equiprobable – в начале моделирования и при каждом убытии сущности блок в случайном порядке осуществляет выбор порта для отбытия следующей прибывающей сущности. Все выходные порты для сущностей одинаково желательны для выбора. Инициализация процесса генерации случайного числа определяется параметром **Initial Seed** (значение неотрицательное целое число).

First port that is not blocked – при попытке приема сущности блок пытается для выпуска сущности использовать порт OUT1. Если этот порт блокирован, то производится попытка использовать блок OUT2 и т.д. Если блокированы все выходные порты, то входной порт блока становится недоступным.

From signal port p – при выборе этого параметра на блоке появляется дополнительный сигнальный порт p. Сигнал на этом порту должен принимать целочисленные значения в диапазоне от 1 до значения

параметра **Number of entity output ports**. При изменении значения сигнального порта выбирается соответствующий выходной порт для следующих прибывающих сущностей.

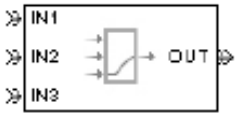
`From attribute` – прибывающая сущность покидает блок через выходной порт, соответствующий значению атрибута, имя которого указано в параметре **Attribute name**. Значение атрибута должно иметь целочисленные значения в диапазоне от 1 до значения параметра **Number of entity output ports**. Если указанный выходной порт блокирован, то блок не принимает сущности до разблокирования соответствующего выходного порта.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке `Statistics` в панели свойств блока. Если отмечен соответствующий пункт, то у блока появляется новый сигнальный выходной порт, на который выводится соответствующая статистическая информация:

`#d` – число сущностей покинувших блок после начала моделирования;

`last` – индекс входного порта, который был доступен в момент прибытия последней сущности. Начальное значение 0.

Блок `Input Switch` позволяет выбрать в процессе моделирования один из нескольких входных портов для сущностей. После выбора одного из входных портов, остальные порты становятся недоступными.



Количество входных портов определяется параметром **Number of entity input ports**, входной порт, которые выбирается для приема сущностей определяется в соответствии с параметром **Switching Criteria**:

`Round robin` – в начале моделирования выбирается порт `IN1`. При каждом следующем прибытии сущности выбирается следующий по порядку входной порт. После выбора последнего порта снова выбирается порт `IN1`.

Equiprobable – в начале моделирования и каждого прибытия сущности блок в случайном порядке осуществляет выбор порта для следующей прибывающей сущности. Все входные порты для сущностей одинаково желательны для выбора. Инициализация процесса генерации случайного числа определяется параметром **Initial Seed** (значение неотрицательное целое число).

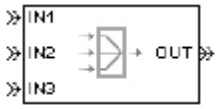
From signal port p – при выборе этого параметра на блоке появляется дополнительный сигнальный порт p. Сигнал на этом порту должен принимать целочисленные значения в диапазоне от 1 до значения параметра **Number of entity input ports**. При изменении значения сигнального порта выбирается соответствующий входной порт для следующих прибывающих сущностей.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке Statistics в панели свойств блока:

#d – число сущностей покинувших блок после начала моделирования;

last – индекс входного порта, который был доступен в момент прибытия последней сущности. Начальное значение 0.

Блок Path Combiner позволяет объединить несколько путей в один.



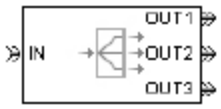
Блок принимает сущности через один из нескольких входных портов и выпускает их через единственный выходной порт. Количество входных портов определяется параметром **Number of entity input ports.**

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке Statistics в панели свойств блока:

#d – число сущностей покинувших блок после начала моделирования;

last – индекс входного порта, который был доступен в момент прибытия последней сущности. Начальное значение 0.

Блок Replicate выпускает через свои выходные порты копии прибывающей в блок сущности.



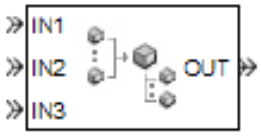
Количество копий, создаваемых блоком, определяется параметром **Number of entity output ports**.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке Statistics в панели свойств блока. Если отмечен соответствующий пункт, то у блока появляется новый сигнальный выходной порт, на который выводится соответствующая статистическая информация:

#a – число сущностей прибывших в блок после начала моделирования;

#d – число сущностей покинувших блок после начала моделирования;

Для объединения нескольких одновременно прибывающих сущностей в одну используется блок Entity Combiner из раздела Entity Management.



Сущности, одновременно прибывающие в блок через входные порты, называются компонентные сущности. Они могут использоваться для представления различных частей некоторого агрегата.

Количество входных портов блока определяется параметром **Number of entity input ports**.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке Statistics в панели свойств блока:

#d – число сущностей покинувших блок после начала моделирования;

Для завершения путей сущностей используется блок `Entity Sink`.

Параметры блока:

`Input port available for entity arrivals` – если выбран этот параметр, то блок принимает прибывающие сущности, в противном случае – блок сущности не принимает, а при попытке прибытия сущности выдается сообщение об ошибке.

`Report number of entities arrived, #a` – при выборе параметра у блока появляется сигнальный порт `#a`, на котором после каждого прибытия сущности выдается информация о количестве принятых блоком сущностей. Начальное значение сигнала после начала моделирования до первого обновления блока равно 0.

3. Моделирование случайных событий.

При работе сборочного участка после сборки изделия возможно появление с вероятностью 30% бракованных изделий. Оценку качества собранных изделий целесообразно производить в блоке модели, имитирующем случайное событие.

Для моделирования случайных событий на компьютерах используются программные генераторы случайных чисел. Все методы и приемы компьютерного моделирования случайных факторов основаны на использовании случайных чисел, равномерно распределенных на интервале [0,1].

Моделирование случайного события A, вероятность которого равна $P(A) = P_c$ Для моделирования случайного события A, вероятность которого равна $P(A) = P_c$, достаточно сформировать одно число r, равномерно

распределенное на интервале [0,1]. При попадании r в интервал [0, P_c] считают, что событие A наступило, в противном случае – не наступило.

Для реализации этого метода можно использовать следующую схему Simulink:

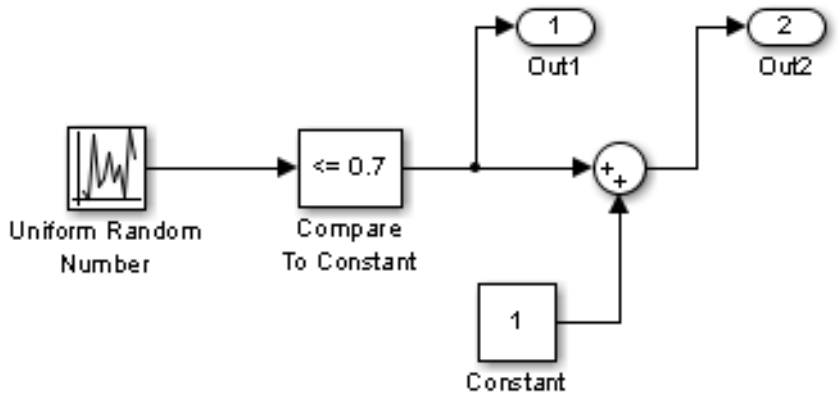


Рис. 2.3. Модель подсистемы `Simulink` для оценки брака.

В схеме используются следующие блоки:

`Uniform Random Number` (равномерно распределенное случайное число) из раздела `Sources`, параметры **Minimum** – 0, **Maximim** – 1.

`Compare To Constant` (равномерно распределенное случайное число) из раздела `Logic and Bit Operations`, параметр **Constant Value** – 0,7.

Модель, представленную на рисунке 2.3, целесообразно использовать в имитационной модели сборочного участка в виде подсистемы. Для преобразования схемы в подсистемы нужно выделить все блоки на рисунке 2.3, нажать правую кнопку мыши и в появившемся контекстном меню выбрать `Create Subsystem from Selection` (создать подсистему из выбранного)

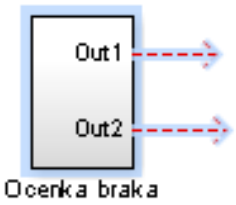


Рис. 2.4. Подсистема для оценки качества собираемых изделии.

На выходе Out1 подсистемы подается 1, если изделие не является бракованным. На выход Out2 подается 1, если деталь бракованная, и 2 – если деталь качественная. Сигнал с выхода 2 можно использовать для управления потоком сущностей через блок `Output Switch`, который моделирует разделение потока изделий на качественные и бракованные.

Для синхронизации работы подсистемы оценки качества с событийно управляемой имитационной моделью целесообразно оформить ее в виде функционально вызываемой подсистемы (`Function-Call Subsystem`). Заготовка для `Function-Call Subsystem` находится в разделе `Ports&Subsystems` библиотеки Simulink. Работа

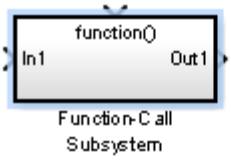
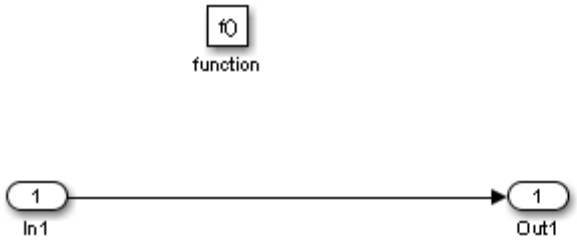


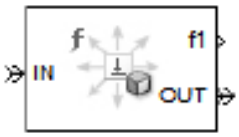
Рис. 2.5. Функционально вызываемая подсистема.

Срабатывание `Function-Call` подсистемы аналогично вызову функции в языках программирования. Подсистема срабатывает в момент прихода специального управляющего сигнала (`function call`) на вход `function()` подсистемы:



При двойном щелчке мышью на иконке блока раскрывается поле для ввода блоков Simulink. При необходимости в подсистему можно добавлять любое количество входов (блоки In раздел Sources) и выходов (блоки Out раздел Sinks).

Для генерации управляющих сигналов для Function-Call Subsystem можно использовать блок Entity Departure Function-Call Generator из раздела Generators/Function-Call Generators библиотеки SimEvents.



Этот блок выдает Function-Call сигнал на выход f1 при каждом убытии сущности из блока.

4. Описание используемых блоков.

Time-Based Entity Generator (Раздел Generators/Entity Generator.) – блок генерирует сущности в моменты времени, определяемые входным сигналом или статистическим распределением.

Порядок генерирования определяется значением параметра **Generate entities upon**:

Intergeneration time from dialog – моменты времени генерации определяются в зависимости от параметров в диалоговых полях блока;

Тип распределения моментов генерации определяется полем **Distribution**. Возможные значения:

- Constant (постоянное) – постоянное время между генерируемыми событиями задается в **поле Period**.
- Uniform (равномерное) – случайное равномерное распределение задается диапазоном в полях **Minimum** и **Maximum**.
- Exponential – экспоненциальное распределение задается параметром **Mean** (среднее).

При установке параметра распределения в значения Uniform или Exponential в диалоговом окне имеется также параметр **Initial seed**, определяющий генерируемый набор случайных чисел. Для фиксированного значения этого параметра случайная последовательность при следующем запуске модели повторится. Типично, значение этого параметра устанавливается в большое (например, пятизначное) нечетное число.

Intergeneration time from port t – моменты времени генерации определяются через сигнальный порт t, информация с которого считывается при старте моделирования и в каждый момент генерации новой сущности. Сигнал должен быть событийным (event-based). Если выставлено это значение, то у блока появляется дополнительный сигнальный порт t.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке **Statistics** в панели свойств блока. Если отмечен соответствующий пункт, то у блока появляется новый сигнальный выходной порт, на который выводится следующая статистическая информация в соответствии:

`#d` – число сущностей покинувших блок после начала моделирования;

`w` – среднее время между генерациями сущностей.

`FIFO Queue` – блок одновременно хранит до `N` сущностей, где `N` – значение параметра **Capacity** (Емкость). Блок пытается выпустить сущность через выходной порт `OUT`, однако если порт `OUT` блокирован, то сущность остается в блоке. Если в блоке хранятся несколько сущностей, то сущности покидают блок в соответствии с дисциплиной первый вошел – первый вышел (first in – first out (FIFO)). Если блок уже хранит `N` сущностей, то входной порт `IN` блока не доступен.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке **Statistics** в панели свойств блока. Если отмечен соответствующий пункт, то у блока появляется новый сигнальный выходной порт, на который выводится следующая статистическая информация:

`#d` – число сущностей покинувших блок через порт OUT после начала моделирования;

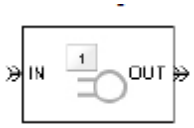
`#n` – число сущностей в очереди;

`w` – среднее время ожидания в этом блоке для всех сущностей, покинувших блок через любой порт;

`len` – среднее число сущностей в очереди по времени, то есть средний по времени сигнал #n.

`Single Server` – блок обслуживает одновременно одну сущность за некоторый интервал времени и затем пытается выпустить сущность через

выходной порт OUT. Если порт OUT блокирован, то сущность остается в блоке до тех пор пока выходной порт не разблокируется.



Время обслуживания (Service Time) определяется через параметры, атрибуты или сигналы в зависимости от значения параметра **Service Time From**. Блок определяет время обслуживания сущности при ее поступлении. Времена обслуживания определяются в секундах.

Значения параметра **Service Time From**:

Dialog – значения времени обслуживания задаются в поле параметра **Service Time**;

Attribute – значения времени обслуживания задаются атрибутом, имя которого указано в поле параметра **Attribute name**;

Signal port t – значения времени обслуживания задаются через сигнальный порт t. Сигнал должен быть событийным (event-based). Если выставлено это значение, то у блока появляется дополнительный сигнальный порт t.

Для сбора статистики блока нужно отметить галочкой нужные сигналы на вкладке **Statistics** в панели свойств блока. Если отмечен соответствующий пункт, то у блока появляется новый сигнальный выходной порт, на который выводится следующая статистическая информация:

#d – число сущностей покинувших блок через порт OUT после начала моделирования;

#n – число сущностей в сервере, 0 или 1;

w – среднее время ожидания в этом блоке для всех сущностей, покинувших блок;

util – утилизация сервера, то есть доля времени моделирования, использованная на хранение сущностей. В начале моделирования утилизация равна 0 или 1 в зависимости от того хранит ли сервер сущность.

Для завершения путей сущностей используется блок `Entity Sink`.

Параметры блока:

Input port available for entity arrivals – если выбран этот параметр, то блок принимает прибывающие сущности, в противном случае – блок сущности не принимает, а при попытке прибытия сущности выдается сообщение об ошибке.

Report number of entities arrived, #a – при выборе параметра у блока появляется сигнальный порт `#a`, на котором после каждого прибытия сущности выдается информация о количестве принятых блоком сущностей. Начальное значение сигнала после начала моделирования до первого обновления блока равно 0.

Для вывода результатов моделирования в виде графиков используется блок `Signal Scope` из раздела библиотеки `SimEvents Sinks`.

Блок создает график, используя данные из событийного сигнала. Данные, откладываемые по вертикальной оси, берутся из сигнала, связанного с входным сигнальным портом `in`.

Параметры блока:

Plot type – тип графика:

`Stair` – ступенчатый;

`Continuous` – непрерывный. Тип графика не меняет дискретный сигнал на непрерывной, а только определяет способ отображения графика.

Данные для горизонтальной оси определяются параметром **X value from**: `Event time` – выводится график данных с порта `in` по времени

моделирования; `Index` – выводится график данных с порта `in` по их индексу, т.е. первое значение имеет индекс 1, второе – 2 и т.д.

Лабораторная работа № 3.

Моделирование систем массового обслуживания в Stateflow

Инструментальное средство **SimEvents**, существенно облегчая процедуру моделирования систем массового обслуживания (СМО) в **Simulink**, не позволяет выходить за рамки предписанных создателями алгоритмов работы отдельных блоков. Большую гибкость может обеспечить работа с такими системами в **Stateflow**. Компонента **Stateflow** также предназначена для работы с дискретно событийными системами, однако, позволяет явно моделировать состояния блоков и системы и определять логические условия для переходов между состояниями. Рассмотрим на примерах как можно в **Stateflow** создавать модели СМО и проводить с ними вычислительные эксперименты.

Пример 1. Специализированный пост диагностики. Пост представляет собой одноканальную СМО. Число стоянок для автомобилей, ожидающих проведения диагностики, ограничено и равно трем. Если все стоянки заняты, то очередной автомобиль, прибывший на диагностику, в очередь на обслуживание не становится и покидает систему. Поток автомобилей, прибывающих на диагностику, распределен по закону Пуассона и имеет интенсивность 0,85 автомобиля в час. Время диагностики автомобиля распределено по показательному закону и в среднем составляет 1,05 ч. (63 мин). Требуется определить вероятностные характеристики поста диагностики, работающего в стационарном режиме.

`Statefow` диаграмма имитационной модели представлена на рисунке 3.1. Модель имеет два И(AND)-состояния `wait` и `serv`. Первое из них соответствует состоянию очереди автомобиля и имеет ИЛИ(OR) - подсостояния `Zero`, `One`, `Two`, `Three`, `Refuse` (соответственно, 0, 1, 2, 3 автомобиля в очереди и состояние “в обслуживании отказано”). Второе надсостояние `serv`, описывающее работу поста диагностики, имеет ИЛИ-

подсостояния `Wait` и `Work` (ожидание прибытия автомобиля и работа по диагностике). Переход по умолчанию в состояние `Zero` приводит к действиям

`tin = ml('poissrnd(70)'); tv1 = t;` % генерация интервала времени ожидания прибытия первого автомобиля и запуск таймера

Команда `ml(evalString, arg1, arg2, …)` осуществляет вызов команды **Matlab** с именем `evalString` и аргументами `arg1, arg2, ….`

Команда `poissrnd(70)` генерирует случайные числа с распределением Пуассона и параметром λ (интенсивность потока требований).

Символ `t` в моделях **Stateflow** обозначает абсолютное время моделирования, полученное от **Simulink**. Если вы введете свой объект с именем t, то ваше определение перекроет встроенное.

При активации состояния `Zero` выполняется действие `och = 0` (служебной переменной `och` присваиваем значение ноль, что свидетельствует об отсутствии автомобилей, нуждающихся в диагностике).

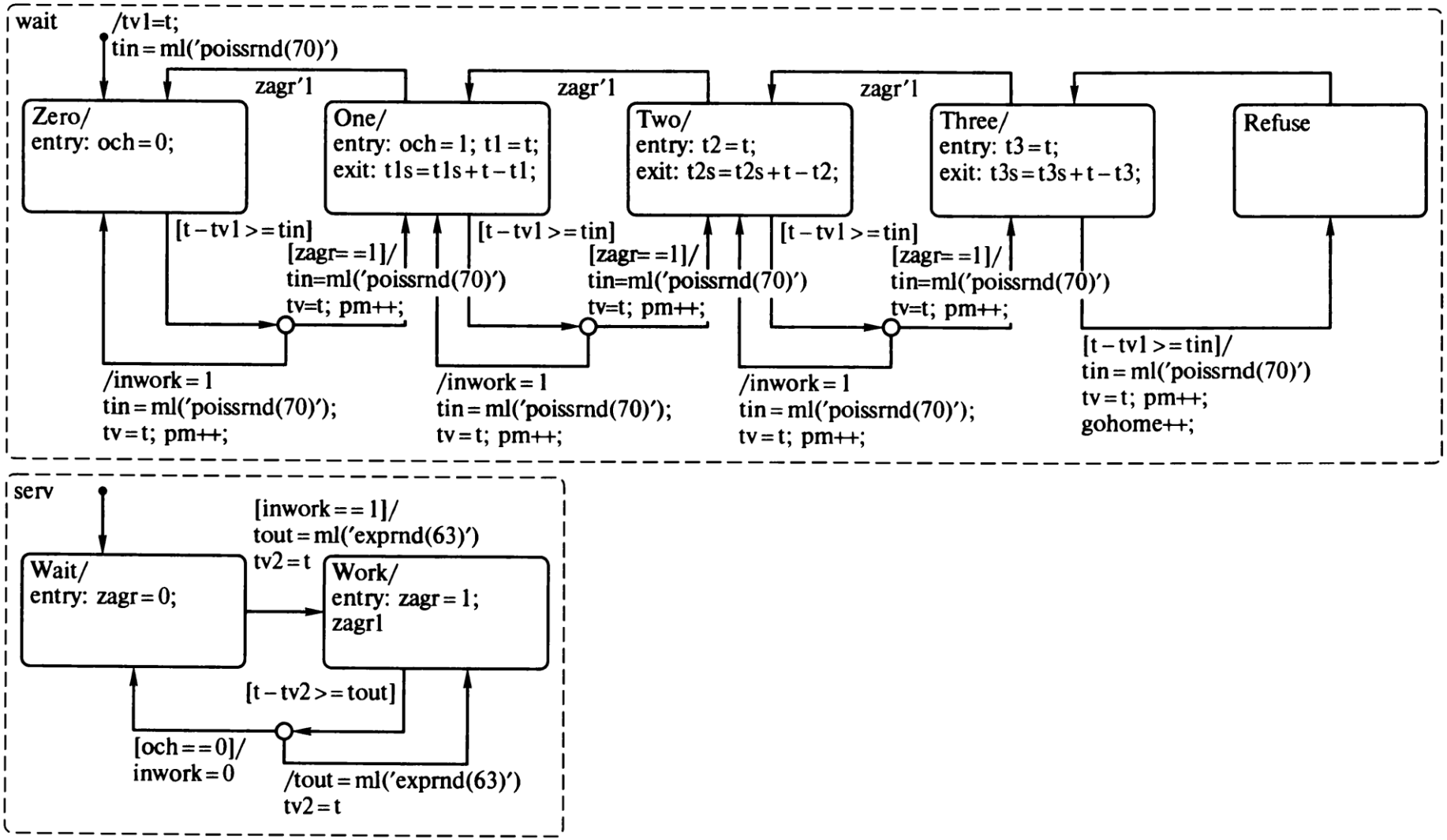


Рис. 3.1. Stateflow диаграмма поста диагностики

Одновременно во втором надсостоянии становится активным подсостояние Wait. При активизации этого состояния выполняется действие zagr = 0 (служебной переменной zagr присваивается значение ноль, это означает, что пост диагностики не занят).

Переход в надсостоянии wait из подсостояния i (i = 0, 1, 2) в обозначаемое кружком квазисостояние (подключаемое соединение или точка принятия решения) происходит при выполнении условия

```
[t – tv1 >= tin]
```

Далее возможны два варианта: если справедливо условие [zagr == 1], то из подключаемого соединения переходим в состояние i+1, при этом выполняются действия

```
tin = ml('poissrnd(70)'); tv1 = t; pm++
```
% генерация интервала времени ожидания прибытия следующего автомобиля, перезапуск таймера и увеличение на 1 счетчика поступивших машин

Если условие [zagr == 1] не справедливо, то из подключаемого соединения возвращаемся в состояние i, при этом дополнительно выполняется действие inwork = 1 (создаем условие для перехода поста диагностики в состояние Work). Другими словами, при поступлении очередной машины определяем, занят ли пост диагностики. Если да – то очередь увеличивается на 1 машину (при i = 0, 1, 2 в ней еще есть свободное место). Если нет – то очередь не увеличивается, но состояние поста диагностики меняется на занятое.

При i = 3 напрямую попадем в состояние Refuse (в обслуживании отказано), а оттуда на следующем такте в состояние Three (три машины в очереди). Показание счетчика числа машин, которым отказано в обслуживании, увеличивается на 1 (действие gohome++;).

Переход в надсостоянии serv из подсостояния Wait в подсостояние Work происходит при выполнении условия [inwork == 1] и вызывает действия

tout = ml('exprnd(63)'); tv2 = t; % генерация времени обслуживания автомобиля и запуск таймера.

При активации этого состояния выполняется действие zagr = 1; (служебной переменной zagr присваивается значение 1, это означает, что пост диагностики занят) и генерируется событие zagr1.

Это событие приводит к переходу в надсостоянии wait из подсостояния i (i = 1, 2, 3) в подсостояние i-1. Переход в надсостоянии serv из подсостояния Work в квазисостояние "подключаемое соединение" происходит при выполнении условия

[t - tv2 >= tout] (время обслуживания истекло).

Далее возможны два варианта: если справедливо условие [och == 0], то из подключаемого условия переходим в состояние Wait. В противном случае – возвращаемся в состояние Work, при этом выполняются действия

/tout = ml('exprnd(63)'); tv2 = t; % генерация времени обслуживания автомобиля и перезапуск таймера.

Другими словами, при освобождении поста диагностики определяем, есть ли машины в очереди. Если да – то пост начинает обслуживать очередную машину. Если нет – то пост ожидает прибытия следующей машины.

Важное значение при работе Stateflow машины имеет порядок обработки параллельных состояний на каждом уровне иерархии и переходов, начинающихся в одном и том же состоянии.

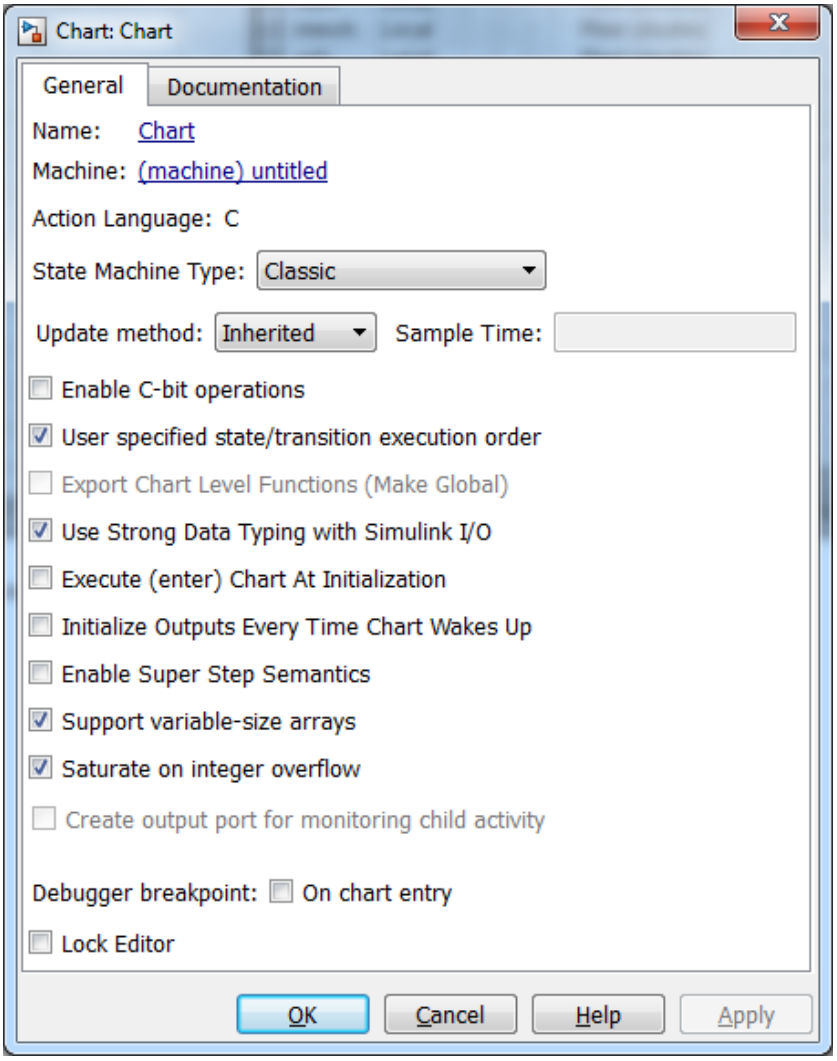


Рис. 3.2. Свойства диаграммы.

Для того, чтобы определить явный порядок обработки переходов и состояний необходимо нажать правую кнопку мыши на свободном месте диаграммы и в появившемся меню выбрать пункт **Propeties.** В появившемся на экране диалоговом окне (рис. 3.2) необходимо отметить пункт **User specified syste/transition execution orde**r.

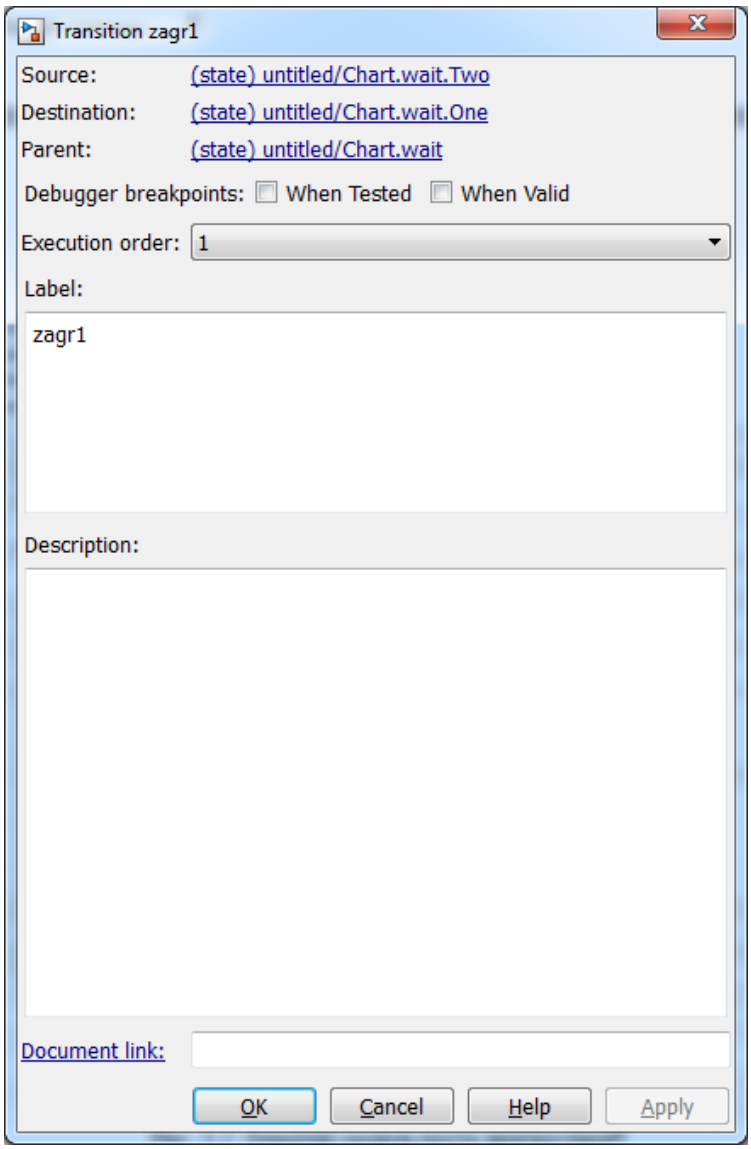


Рис. 3.3. Свойства перехода.

Для задания порядка обработки переходов нужно выделить переход, нажать еа нем правую кнопку мыши и и в появившемся меню выбрать пункт **Propeties.** В появившемся на экране диалоговом окне (рис. 3.3) порядок определяется в списке **Execution Order**. Аналогично задается порядок обработки параллельных состояний.

Если выбран неявный порядок обработки, то состояния обрабатываются в порядке их создания. Переходы, начинающиеся в одном и том же состоянии, обрабатываются в следующем порядке.

1. В порядке уменьшения иерархии перехода.

2. Переходы на одном уровне иерархии обрабатываются в следующем порядке:

а) переходы, управляемые событиями и логическими условиями;

б) переходы, управляемые событиями;

в) переходы, управляемые логическими условиями;

г) безусловные переходы;

3. Если в соответствии с вышеупомянутыми правилами переходы имеют одинаковый приоритет, то они обрабатываются в соответствии с направлением. Наименьший приоритет имеет переход с направлением на 12 часов. Далее приоритет возрастает по направлению часовой стрелки.

В рассмотренном примере порядок обработки параллельных следующий: сначала обрабатывается очередь (состояние `wait`), затем пост диагностики (состояние `serv`). По два перехода начинаются в состояниях `One`, `Two`, `Three`. Здесь сначала проверяется условие перехода, определяемого событием `zagr1`.

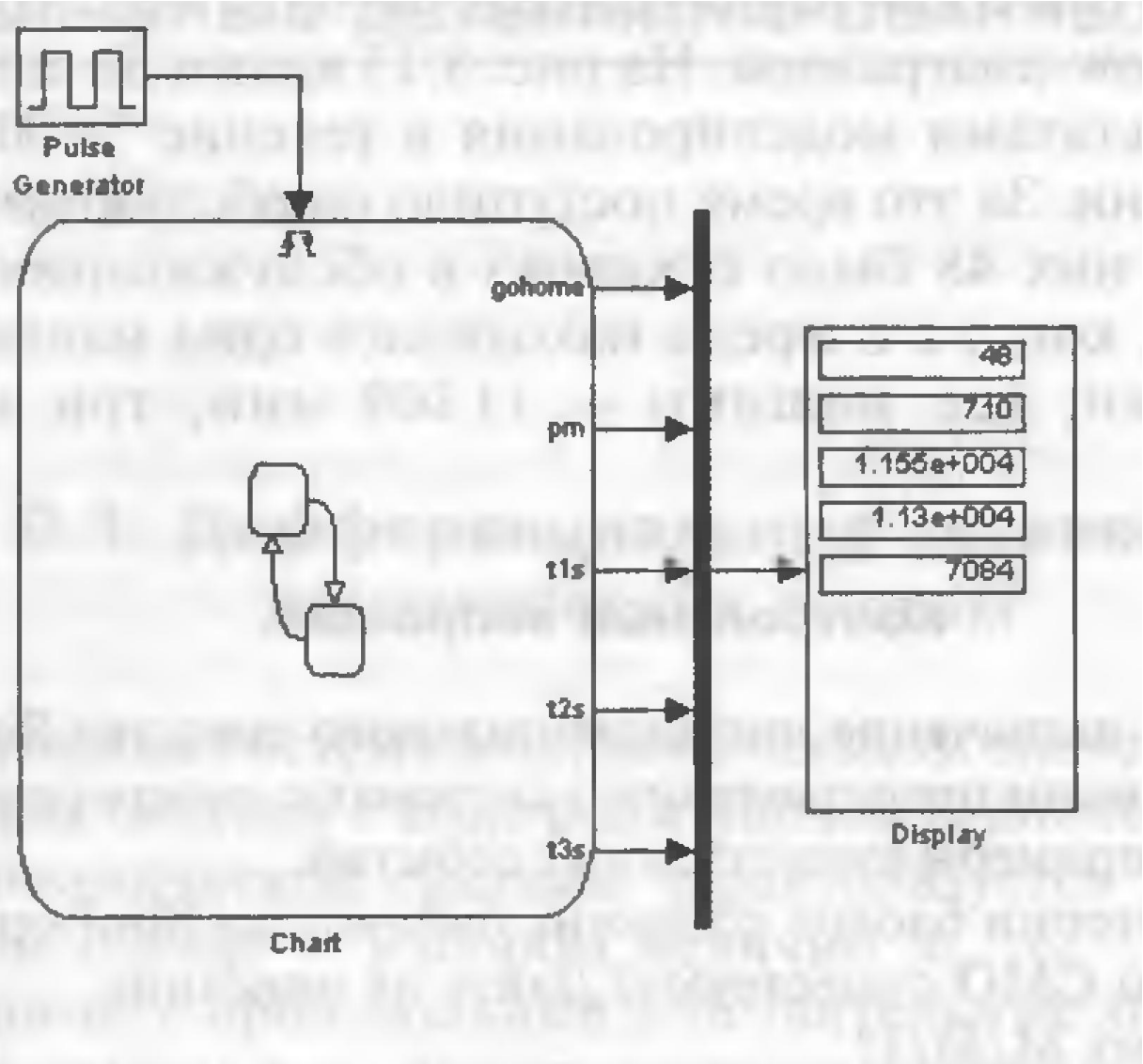


Рис. 3.4. Simulink-модель поста диагностики

На рисунке 3.4 показана модель `Simulink` с результатами моделирования в течение 50 000 мин модельного времени. За это время поступило на обслуживание 710 автомобилей, из них 48 было отказано в обслуживании. Длительность периода, когда в очереди находилась одна машина, равняется 11 550 мин, две машины – 11 300 мин, три машины – 7 084 мин.

В модели `Simulink` блок `Pulse Generator` моделирует изменение модельного времени в имитационной модели. Настройки блока: амплитуда – 1, период – 1 сек.

В модели определено два события:

локальное zagr1;

входное (Input From Simulink) tm , тип триггера – Either.

Выходные переменные модели: gohome, pm, t1s, t2s, t3s.

Остальные переменные (inwork, och, t1, t2, t3, tin, tout, tv1, tv2, zagr) – локальные.

Пример 2. В рассмотренном выше примере в очереди выделены состояния, соответствующие различному количеству ожидающих автомобилей. Это позволяет детально исследовать вероятностные характеристики очереди. Если в такой детализации нет необходимости то очередь можно существенно упростить (рис.3.5).

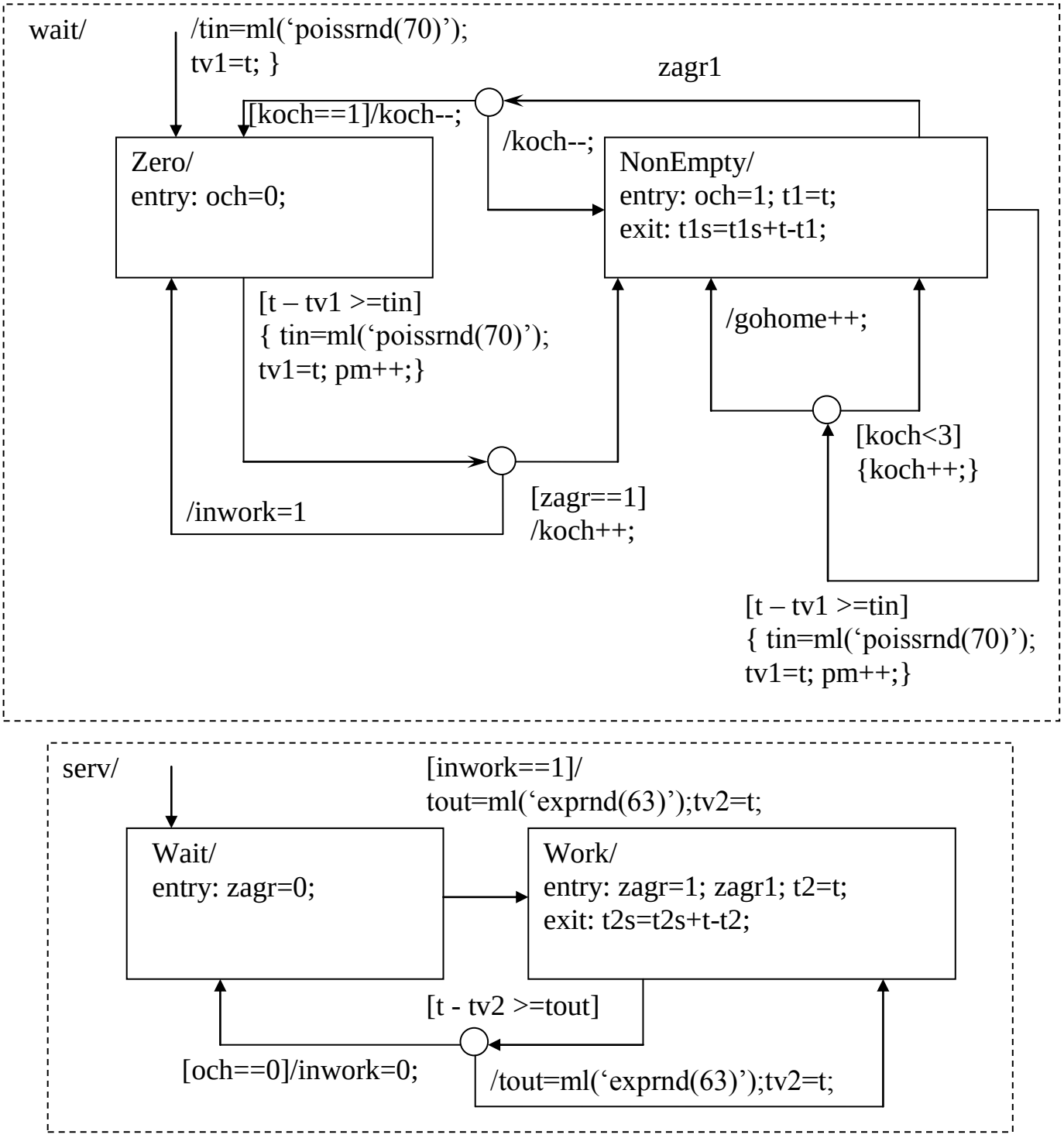


Рис. 3.5 `Stateflow` диаграмма модифицированной модели поста диагностики

В данной модели переменная t2s используется для расчета времени диагностики автомобилей. Введена дополнительная локальная переменная koch для подсчета числа автомобилей в очереди на диагностику.

Задание. Проведите моделирование по примерам 1 и 2

Моделирование системы обработки информации в Stateflow

Постановка задачи. Система обработки информации содержит мультиплексный канал и три миниЭВМ. Сигналы от датчиков поступают на вход канала через интервалы времени 5-15 мкс. В канале они буферизуются и предварительно обрабатываются в течении 7-13 мкс. Затем они поступают на обработку в ту миниЭВМ, где имеется наименьшая по длине входная очередь. Емкости входных накопителей во всех миниЭВМ рассчитаны на хранение величин 10 сигналов. Время обработки сигнала в любой миниЭВМ равно 33 мкс.

Цель исследования: Смоделировать процесс обработки 500 сигналов, поступающих с датчиков. Определить средние времена задержки сигналов в канале и миниЭВМ и вероятности переполнения входных накопителей.

На приведенном ниже рисунке 4.1 представлена структурная схема системы.

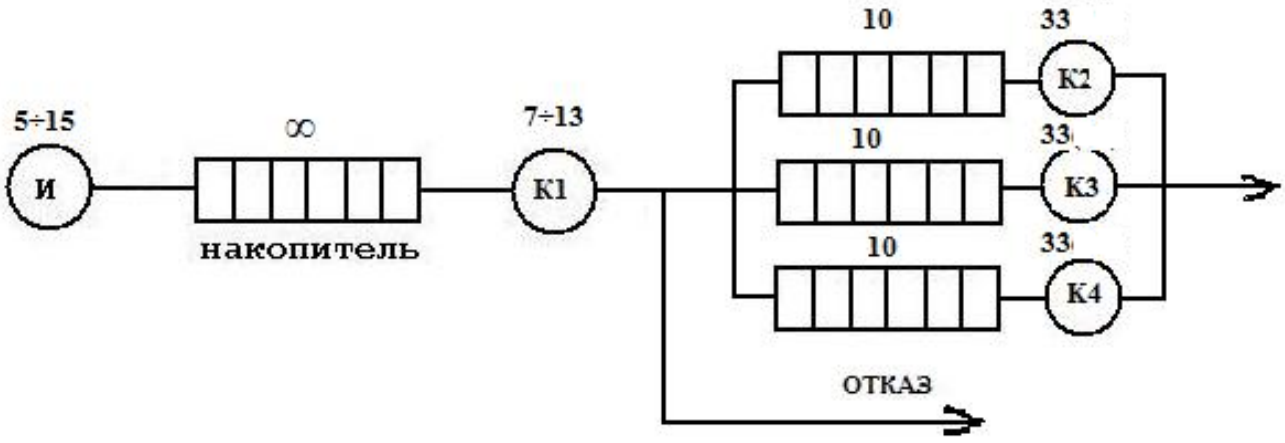


Рисунок 4.1 – Структурная схема системы

Имитационная модель характеризуется набором переменных, с помощью которых удается управлять изучаемым процессом, и набором начальных условий, когда можно изменять условия проведения машинного эксперимента. Для полного анализа характеристик процесса

функционирования систем приходится многократно воспроизводить имитационный эксперимент, варьируя исходные данные задачи. При этом, как следствие, возникает увеличение затрат машинного времени. Общий алгоритм моделирования системы представлен на рисунке 4.2.

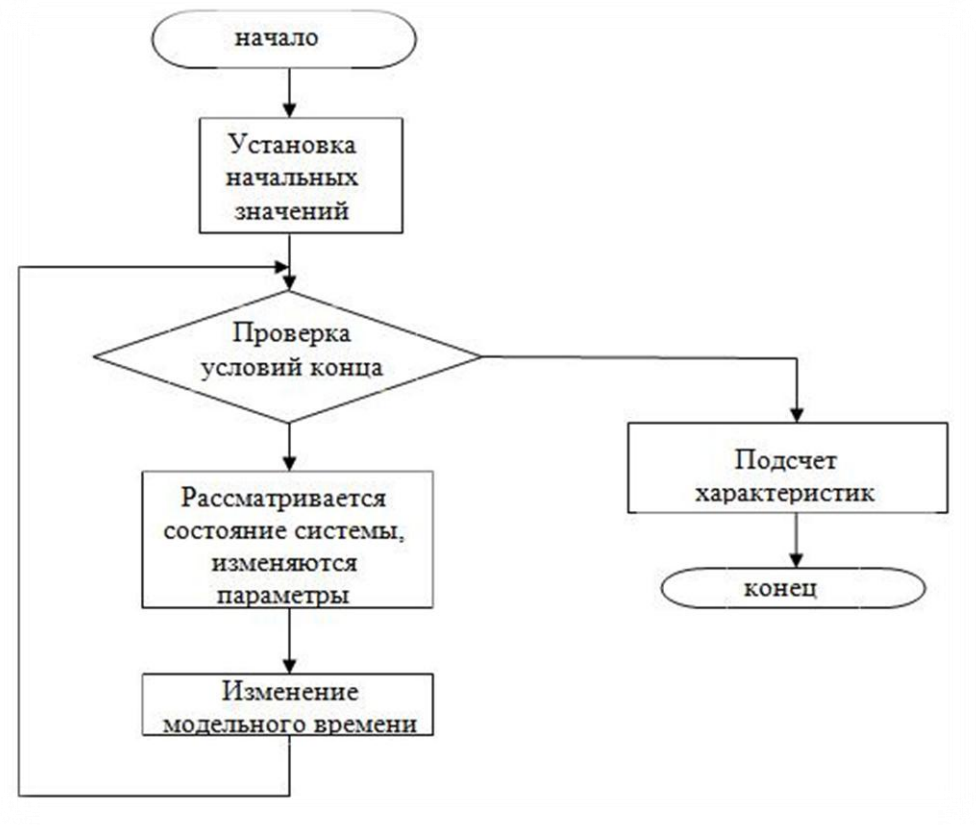


Рисунок 4.2 – Общий алгоритм моделирования системы

Задание. Промоделируйте рассмотренную выше систему обработки информации в системе **Stateflow**. Блоки структурной схемы модели моделируются параллельными (AND) блоками (состояниями) **Stateflow.**

Перечень блоков в порядке их обработки:

- 1 – блок генерации заявок и бесконечная очередь;
- 2 – канал предварительной обработки заявок;
- 3 – блок распределения заявок по 3-м накопителям;
- 4, 5, 6 – мини-ЭВМ 1,2,3 обработки заявок;

В модели используются следующие переменные.

Внутренние переменные:

- bocher – бесконечная очередь;
- kolzajv – количество сгенерированных заявок;
- osv_kan – переменная флаг, показывающая состояние канала предварительной обработки (занят-1/свободен-0);
- n1 – переменная, показывающая количество заявок в накопителе 1;
- n2 – переменная, показывающая количество заявок в накопителе 2;
- n3 – переменная, показывающая количество заявок в накопителе 3;
- Tobs – время обслуживания заявки в канале предварительной обработки;
- Tobs1 – время обслуживания заявки в ЭВМ 1;
- Tobs2 – время обслуживания заявки в ЭВМ 2;
- Tobs3 – время обслуживания заявки в ЭВМ 3;
- Tobssum – суммарное время, затраченное на обслуживание заявок каналом предварительной обработки;
- Tobs1sum – суммарное время, затраченное на обслуживание заявок ЭВМ_1;
- Tobs2sum – суммарное время, затраченное на обслуживание заявок ЭВМ_2;
- Tobs3sum – суммарное время, затраченное на обслуживание заявок ЭВМ_3;
- raspr – переменная, служащая для запоминания распределения;

Выходные переменные для связи с Simulink:

- Total_time – общее время работы системы;
- ver_per – вероятность переполнения входных накопителей;

- kol_otkaz – количество отказов;
- kol_z1 – количество заявок обслуженных ЭВМ 1;
- kol_z2 – количество заявок обслуженных ЭВМ 2;
- kol_z3 – количество заявок обслуженных ЭВМ 3;
- svzsevm1 – среднее время задержки сигналов в ЭВМ 1;
- svzsevm2 – среднее время задержки сигналов в ЭВМ 2;
- svzsevm3 – среднее время задержки сигналов в ЭВМ 3;
- svzskan – среднее время задержки сигналов в канале;
- st – переменная флаг, показывающая, что процесс нужно завершить (завершение-1/выполнение-0);

Определение времен поступления заявок в систему и времен обслуживания в мультиплексном канале и миниЭВМ 1-3 производится аналогично работе № 3. Дополнительные локальные переменные введите самостоятельно.

Для моделирования случайных чисел равномерно распределенных в интервале `[a,b]` используется команда Matlab

```
ml('round(unifrnd(a,b))')
```

`Simulink`-модель имеет тот же вид, что и в лабораторной работе № 3. В модели Simulink блок `Pulse Generator` моделирует изменение модельного времени в имитационной модели. Настройки блока: амплитуда – 1, период – 1 сек.

Моделирование распределения заявок по трем блоками

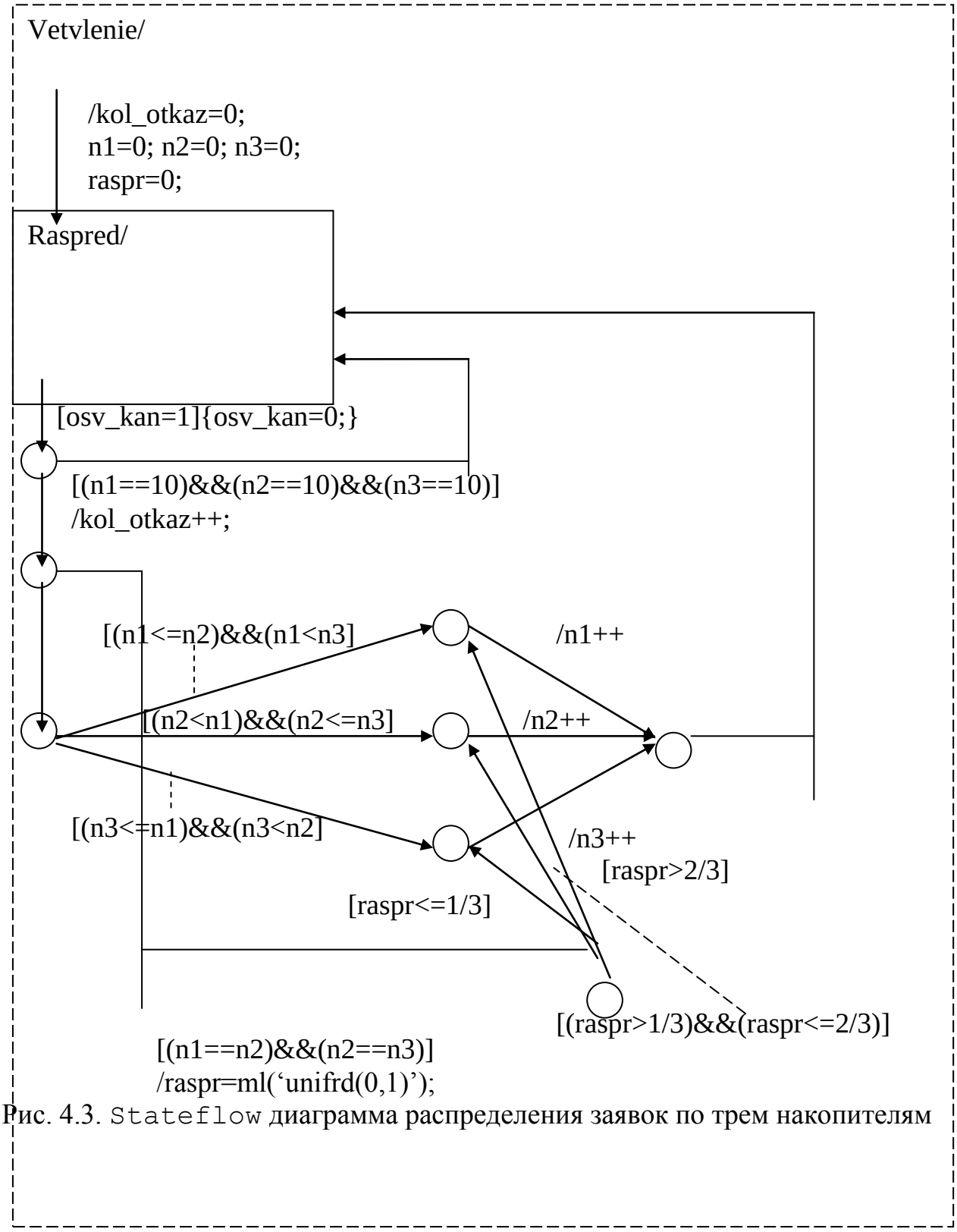
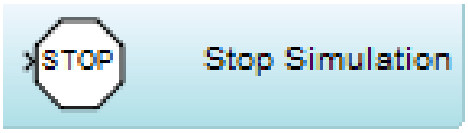


Рис. 4.3. `Stateflow` диаграмма распределения заявок по трем накопителям

Stateflow диаграмма блока рвспределения заявок по трем накопителям представлена на рисунке 4.3. Данный блок анализирует состояния накопителей перед ЭВМ и направляет заявку в накопитель с наименьшей очередью, а также фиксирует наличие отказа в случае, когда все накопители заняты.

Для завершения процесса моделирования можно использовать блок Stop Simulation из раздела библиотеки Sinks:



Блок завершает процесс моделирования при подаче на его вход любого ненулевого сигнала.

При достижении количества заявок (переменная kolzajv) заданного числа (500) необходимо сгенерировать сигнал st=1, означающий, что моделирование требуется завершить, и вычислить итоговые характеристики системы.

Средние времена задержки в каналах предварительной обработки и трех миниЭВМ определяется как отношение соответствующих суммарного времени обработки в канале к количеству обработанных заявок.

Вероятность переполнения входных накопителей можно оценить как отношение количества отказов к общему количеству поступивших заявок.

Соответствующие вычисления можно выполнить в блоке генерации заявок.

Лабораторная работа № 5.

Совместное использование блоков SimEvents, Stateflow и Simulink в событийно-управляемых моделях.

Совместное использование блоков SimEvents и Stateflow.

Для моделирования дискретно-событийных систем используются компоненты SimEvents и Stateflow. Однако при моделировании они играют разные роли:

- Блоки SimEvents используются для моделирования движения сущностей через систему и позволяют исследовать связь движения сущностей с функционированием системы. Сущности при движении могут перемещать данные. Кроме того, блоки SimEvents могут генерировать события в моменты времени, полностью независимые от временных шагов, определяемых программным обеспечением Simulink.
- Диаграммы Stateflow используются для моделирования состояния блока или системы. На диаграммах показаны возможные значения состояний и описывают условия, определяющие переходы между состояниями. Визуализация работы диаграммы Stateflow показывает переходы между состояниями, но не показывает перемещение данных.

Подсистема Stateflow позволяет реализовать переходы между конечным числом состояний сервера. Если необходимо реализовать моделирование более двух состояний, то использование диаграммы Stateflow более удобно и естественно по сравнению с комбинацией блоков Enabled Gate и логических блоков. Для обеспечения реакции блоков Stateflow на асинхронные изменения состояния можно

использовать сигналы function call. Для генерирования сигналов function call в ответ на сигнальные события или поступления сущностей можно использовать блок из разделов Entity Generators и Event Translation. Кроме того, блок Stateflow может генерировать function call-сигнал, который может открыть вентиль, перезапустить счетчик сущностей, или инициировать генерацию новой сущности блоком Entity Generator.

Преобразования событийных сигналов.

При дискретно-событийном моделировании, как правило, используются сигналы, которые изменяются при наступлении событий. Например, число сущностей в сервере является статистическим выходным сигналом из блока сервера. Значение этого сигнала изменяется при прибытии сущности в сервер или убытии сущности из него. *Событийный сигнал* (*event-based signal*) – это сигнал, который может измениться при некотором дискретном событии. Большинство выходных сигналов из блоков SimEvents являются событийными.

В отличие от временных сигналов (*time-based signals*) событийные сигналы:

- не характеризуются дискретной выборкой времен;
- могут обновляться в моменты времени, не соответствующие шагам времени, определяемым временной динамикой модели;
- могут подвергаться многократным изменениям в один момент времени.

Например, рассмотрим сигнал, представляющий количество сущностей в сервере. Вычисление этого сигнала с постоянным временным шагом бесполезно, если никакие сущности не прибывают и не убывают в течение длительных периодов времени. Вычисления с постоянным временным шагом не точны, если сущности прибывают или убывают между периодами

квантования системы, так как при вычислениях эти сущности будут пропущены. Одновременные события делают временной сигнал многозначным. Например, если сущность завершает обслуживание и покидает сервер, что позволяет другой сущности прибыть в сервер в тот же момент времени, то количество сущностей в один и тот же момент времени равно одновременно 0 и 1. Когда вы используете событийный сигнал для управления динамикой моделирования, то понимание того, когда и в какой последовательности блоки обновляют сигналы, и когда другие блоки реагируют на обновляемые значения, приобретает большое значение.

Таким образом, временные и событийные сигналы имеют различные характеристики. Рассмотрим случаи, при которых необходимо преобразовывать временные сигналы в событийные и обратно:

- Если необходимо связать временной сигнал со входом блока SimEvents, то перед входным портом нужно вставить блок преобразования сигнала. Для преобразования используются блоки Timed to Event Signal для сигналов с данными и Timed to Event Function-Call для сигналов вызова функций.

Если необходимо использовать данные из событийного сигнала для воздействия на временную динамику, то перед входным портом нужно вставить блок преобразования сигнала. Для преобразования используются блоки Event to Timed Signal для сигналов с данными и Event to Timed Function-Call для сигналов вызова функций.

Блок Event to Timed Signal из раздела Gateways.

Блок преобразует событийный сигнал во временной. Выходной сигнал предполагает точно одно значение в каждый момент времени. Выходной сигнал почти полностью идентичен входному за исключением:

- В выходном сигнале пропущены сигналы нулевой длительности (если они имеются).
- Выходной сигнал имеет моменты квантования по времени типа “fixed in minor steps”. В результате выходной сигнал может содержать моменты квантования, не связанные с входным сигналом, а с другими временными сигналами модели.
- Выходной сигнал подходит для моделирования временной динамики. Сигнал не может служить входом блока, требующего событийные входные сигналы.
- Начальное выходное значение такое же как и начальное входное значение. Однако, если входной сигнал не определен при T=0, то выходной сигнал блока имеет начальное значение 0.

Обратное преобразование выполняется блоком `Timed to Event Signal`.

Блок `Event to Timed Function-Call` из раздела `Gateways`.

Блок преобразует скалярный событийный сигнал вызова функции во временной. Выходной сигнал почти полностью идентичен входному за исключением того, что выход может служить входным сигналом блока, требующего временной сигнал вызова функции на входе.

Обратное преобразование выполняется блоком `Timed to Event Function-Call`

Если необходимо выполнить вычисления с событийным сигналом, то при этом можно использовать только ограниченной множество блоков. Во-первых, нельзя использовать блоки с непрерывным временем. Во-вторых, в блоках необходимо задать параметр **Sample Time** (время квантования) равным -1 (наследуется от предыдущих блоков).

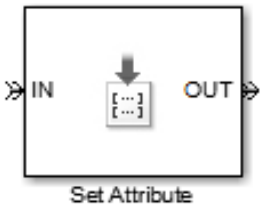
Определение атрибутов сущностей.

К сущностям можно присоединять данные, используя один или несколько *атрибутов* сущности. Каждый атрибут имеет имя и численное значение. Можно считывать или изменять значения атрибутов во время процесса моделирования. Значениями атрибута могут быть вещественные или комплексные числа типа `double`. Кроме того, можно использовать массивы чисел любой размерности. Во время процесса моделирования размерности должны быть фиксированы.

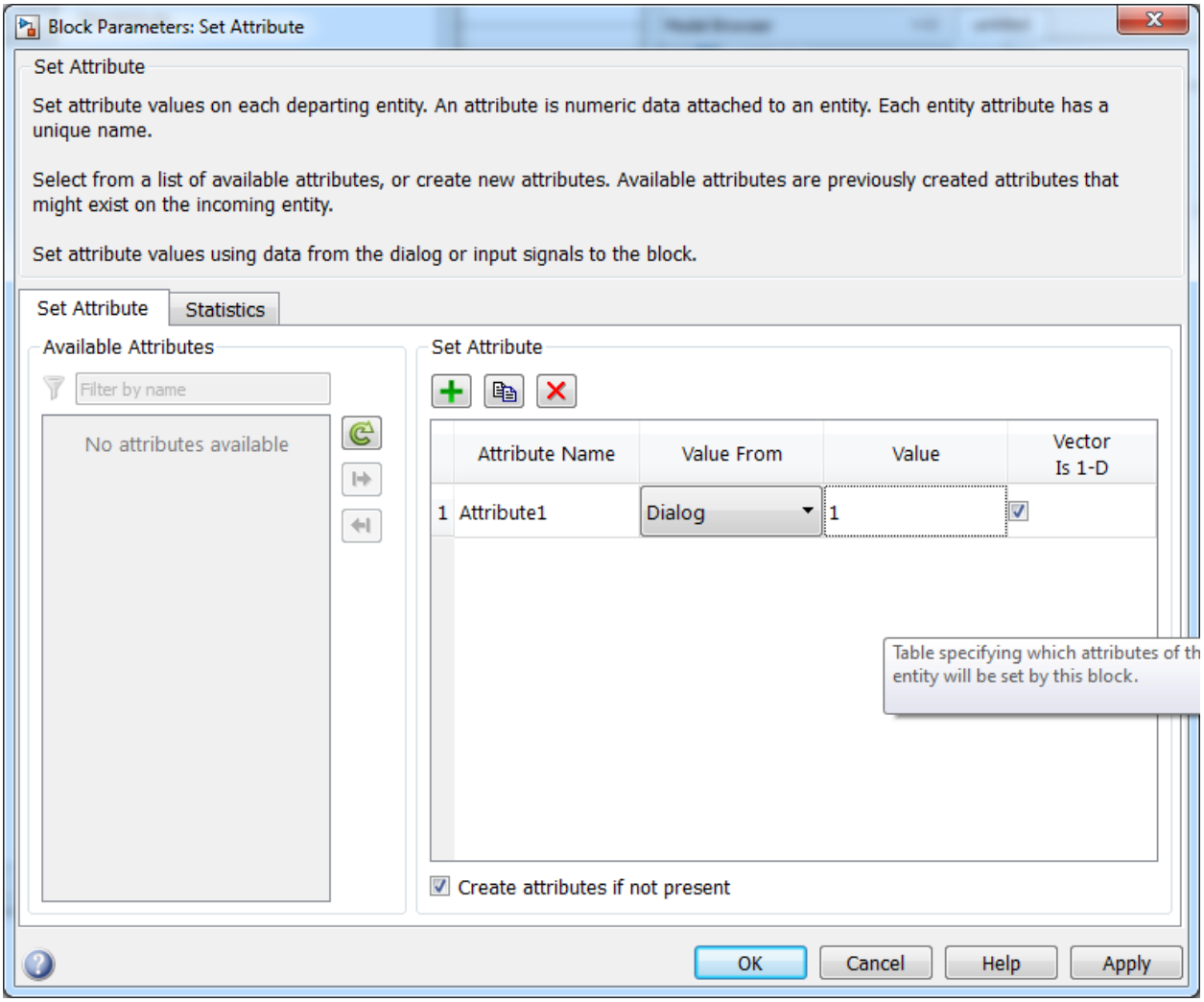
Для добавления атрибута к сущности используется блок `Set Attribute`. Для считывания атрибута из сущности и формирования соответствующего сигнала используется блок `Get Attribute`. Для графического отображения значения атрибута используется блок `Attribute Scope`, в параметре **Y attribute name** которого нужно указать имя атрибута.

Значения атрибутов можно использовать для переключения выходов блоков `Output Switch`, задания времен обслуживания серверов.

Блок `Set Attribute` из раздела `Attributes`.



Блок принимает сущность, присваивает ей данные и выпускает сущность через выходной порт. Присвоенные данные хранятся в атрибутах сущности. Каждый атрибут имеет имя (name) и значение (value). В каждой сущности можно определить до 32 атрибутов.



Имена атрибутов можно задавать непосредственно в колонке **Attribute Name** диалогового окна параметров блока или выбирать из списка доступных атрибутов (**Available Attributes**), который формируется автоматически по входящим в блок путям сущности модели `SimEvents`.

В столбце **Value From** определяется задаются ли значения атрибута в режиме диалога (**Dialog**) в столбце **Value**, либо в процессе моделирования через сигнальный порт (**Signal Port**). В последнем случае к блоку добавляется дополнительный входной порт с именем атрибута.

Завершение процесса моделирования.

При расчете статистических характеристик моделируемой системы очень важно обеспечить завершение процесса в правильное время. Типичные критерии для завершения дискретно-событийного моделирования:

- Достижение заданного времени моделирования.
- Достижение заданного количества обслуженных пользователей, обработанных деталей и т.д.
- Достижение в процессе моделирования заданного состояния, например, переполнения очереди или поломки машины.

Время окончания процесса моделирования задается параметром **Stop Time** в диалоговом окне, вызываемом командой меню **Simulation->Model Configuration Parameters**.

Для завершения процесса моделирования можно использовать блок `Stop Simulation` из раздела библиотеки `Sinks`:

Блок завершает процесс моделирования при подаче на его вход любого ненулевого сигнала.

Для завершения процесса моделирования по достижении заданного числа сущностей можно использовать следующие действия.

1. Многие блоки в качестве выходного сигнала используют количество покинувших блок сущностей. Расчет сигнала определяется параметром **Number of Entity Departed**. Задайте этот выходной сигнал блока, отметив параметр **Number of Entity Departed** в настройках блока

2. Соедините выходной сигнальный порт со входом атомарной подсистемы (`Atomic Subsystem`). Удалите из атомарной подсистемы выходной порт (`Out`).

3. Входной порт подсистемы соедините с блоком `Compare To Constant` (Сравнение с константой).

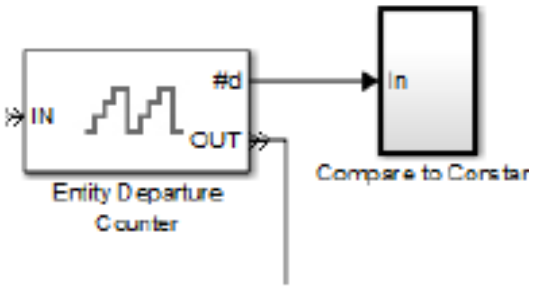
4. В настойках блока `Compare To Constant`:

- задайте оператор >=;
- задайте параметр **Constant value** желаемому значению сущностей;
- задайте в параметре **Output data type mode** значение Boolean;

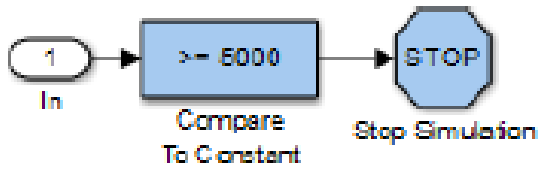
5. Соедините выход блока `Compare To Constant` со входом блока `Stop Simulation`.

Результат имеет следующий вид:

Модель верхнего уровня:



Содержание подсистемы:



Аналогично с помощью блока `Stop Simulation` можно завершать моделирование по достижении некоторого состояния, достижения сущностью конца некоторого пути и т.д.

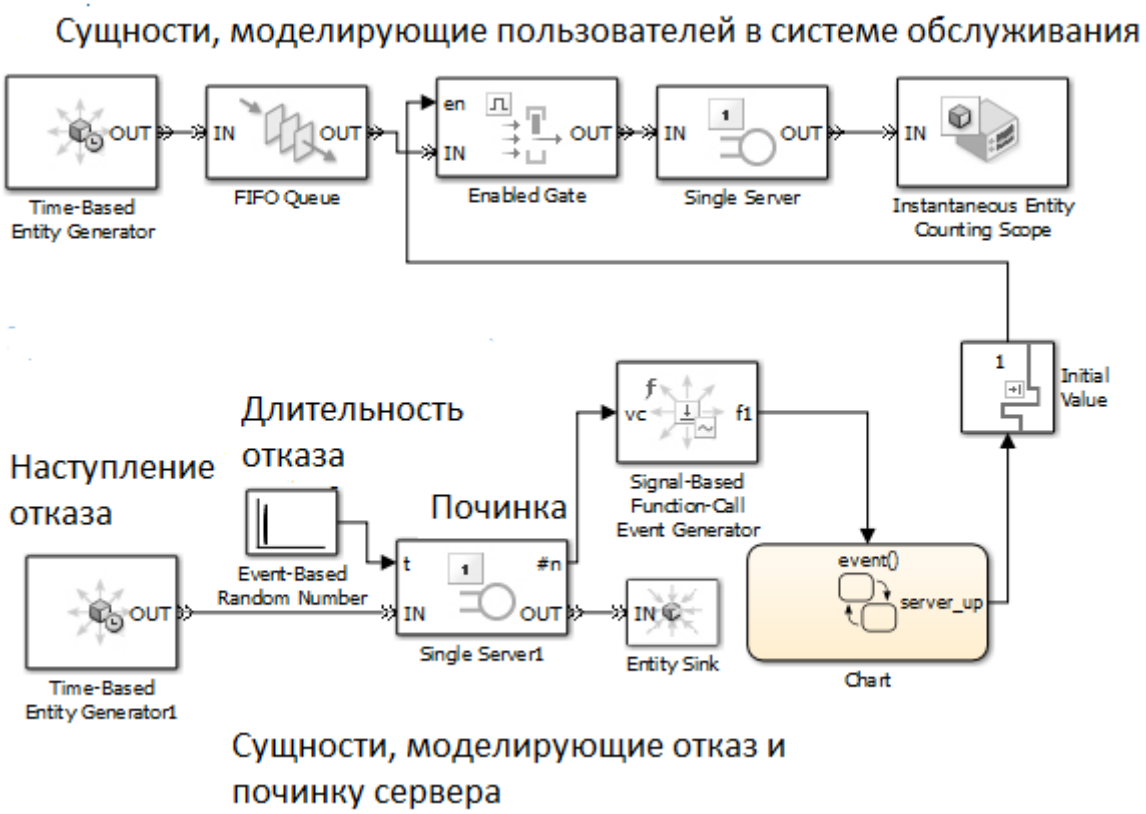
Моделирование отказов машин и оборудования.

В некоторых задачах необходимо моделировать ситуации нарушения работы сервера. Например, прибор может быть сломан и позднее починен, соединение в вычислительной сети может быть нарушено и позднее восстановлено. Такие ситуации удобно моделировать с помощью состояний и переходов между ними. Блоки в разделе `Servers` библиотеки `SimEvents` не имеют встроенных средств для моделирования состояний. Однако можно моделировать состояния сервером с помощью других средств `Simulink` и `Stateflow`.

Для моделирования состояния, представляющего невозможность или отказ сервера принимать сущность даже в случае, если сервер не занят, можно использовать вентиль – блок `Enabled Gate`, предшествующий серверу.

Блок `Enabled Gate` предотвращает доступ к серверу пока управляющий сигнал на входном порту **en** равен нулю или отрицателен. Логика, формирующая сигнал **en**, определяет находится ли сервер в состоянии отказа или поломки. Такую логику можно реализовать с помощью блока `Matlab Function`, подсистемы, содержащей логические блоки, или графа состояний `Stateflow`.

Пример 1. Отказ и восстановление (починка) сервера. В следующем примере блок `Stateflow` использован для описания машины с двумя состояниями. Сервер либо отключен (отказ), либо включен (работа). Состояние сервера определяется выходным сигналом блока Stateflow, который используется для открытия вентиля `Enabled Gate`, предшествующего серверу.

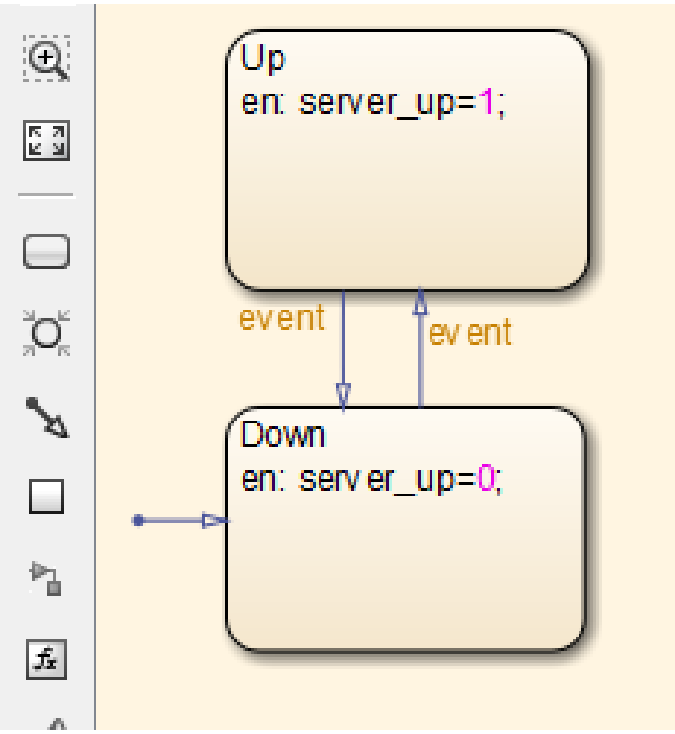


Модель состоит из двух параллельных систем обслуживания. Нижняя система обслуживания моделирует поток отказов. Генерация сущности в этой подсистеме соответствует отказу в верхней системе обслуживания. Обслуживание сущности-отказа (починка) соответствует времени, в течение которого сервер в верхней системе отключен. Завершение обслуживания сущности-отказа соответствует возврату к функционированию верхней системы.

Когда нижняя система генерирует сущность, изменение ее серверного сигнала **#n** включает блок `Stateflow`, определяющий состояние верхней

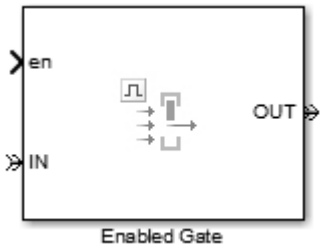
системы обслуживания. Увеличение сигнала **#n** приводит к отключению сервера, а уменьшение – к включению.

Диаграмма `Stateflow` для данного примера представлена на следующем рисунке.



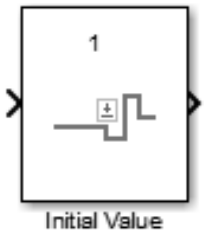
Используемые блоки `SimEvents`.

Блок `Enabled Gate` из раздела `Gates`.



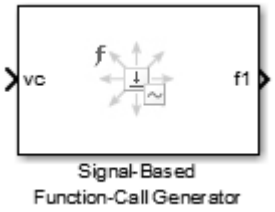
Вентиль позволяет проходить через него сущностям при положительном сигнале на управляющем входе **en**. Вход **en** используется для событийного управляющего сигнала.

Блок `Initial Value` из раздела `Signal Management`.



Блок устанавливает начальное значение для событийного сигнала. До первого временного такта (то есть, до первого обновления сигнала) на входном порту блока выходной сигнал блока равен значению параметра **Value until first sample time hit**. После первого временного такта выходной сигнал равен входному сигналу.

Блок `Signal-Based Function-Call Event Generator` из раздела `Generators/Function-Call Generators`.



Этот блок преобразует сигнальное событие или `function-call` сигнал в один или два `function-call` сигнала, которые можно использовать для вызова `function-call` подсистем, блоков `Stateflow` или других блоков, которые используют `function-call` входы.

Основной критерий, который используется блоком для генерации `function-call` сигнала, задается параметром **Generate function-call upon only**:

`Change in signal from port vc` – критерий удовлетворяется при увеличении или уменьшении событийного сигнала на порту `vc`. Изменение сигнала, которое приводит к появлению выходного сигнала, определяется параметром **Type of change in signal value**: rising/falling/either (подъем/спад/любой).

`Trigger from port tr` – критерий удовлетворяется при появлении на порту `tr` триггерного сигнала. Тип триггера определяется параметром **Trigger type**: rising/falling/either.

`Sample time hit from port ts` – критерий удовлетворяется при обновлении сигнала на порту `ts`.

`Function Call from port fcn` – критерий удовлетворяется при подаче на вход `fcn` событийного `function call` сигнала.

Для генерации второго `function-call` сигнала при каждом событии выберите параметр **Generate optional f2 function call**. Если блок генерирует `function-call` сигналы на двух выходах `f1` и `f2`, то он сначала генерирует сигнал на выходе `f1`, а затем – на выходе `f2`, как последовательные части одной и той же операции.

Вторичные критерии для генерации `function-call` сигналов определяются параметрами **Suppress function call f1 if enable signal e1 is not positive** и **Suppress function call f2 if enable signal e2 is not positive**. Если соответствующий параметр отмечен, то у блока появляются дополнительные управляющие входы `e1` и/или `e2`, при подаче на которых неположительных значений соответствующий `function-call` сигнал подавляется. Порты `e1` и `e2` действуют независимо друг от друга, определяя дополнительные условия для подачи соответствующих `function-call` сигналов.

Задание. Выполните моделирование рассмотренного выше примера 1. Модифицируйте модель путем добавления фазы (состояния) подготовки машины после ремонта. Добавьте соответствующее состояние в диаграмму `Stateflow`.

Блок `Enabled Gate` в верхней системе обслуживания не должен открыться до тех пор, пока не завершаться фазы ремонта и полготовки системы . В нижней системе обслуживания добавьте дополнительный блок для моделирования процесса подготовки машины.

Список литературы.

1. Кельтон В., Лоу А. Имитационное моделирование. Классика CS.3-е изд.-СПб.:Питер;Киев:Издательская группа BHV, 2004.-847 с., ил.

2. Лазарев Ю. Моделирование процессов и систем в MatLab. Учебный курс.-СПб.:Питер; Киев: Издательская группа БХВ,2005.-512 с.:ил.

2. Морозов В.К., Рогачев Г.Н. Моделирование информационных и динамических систем М.: Издательский центр “Академия”, 2011. – 384 с.