МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВПО «РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ (РИНХ)»

Щербаков С.М.

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ ЭКОНОМИЧЕСКИХ ПРОЦЕССОВ В СИСТЕМЕ ARENA

Учебное пособие

г. Ростов-на-Дону
2012

УДК 330.4 (075)+004 (075)
Щ 61

Щербаков, С.М.
Щ 61 Имитационное моделирование экономических процессов в системе Arena: Учебное пособие / С.М. Щербаков ; Рост. гос. эконом. ун-т (РИНХ). – Ростов н/Д, 2012. – 128 с.
ISBN 978-5-7972-1868-5

Учебное пособие предназначено для изучения основ практической работы с одним из наиболее распространенных и мощных инструментов имитационного моделирования Rockwell Arena. Детально описаны возможности системы и принципы работы с нею. Представлены и разобраны примеры решения задач моделирования различного уровня и различной направленности.

Учебное пособие предназначено для студентов всех форм обучения направлений «Прикладная информатика», «Бизнес-информатика».

Утверждено учебно-методическим советом университета

Рецензенты:

Хубаев Г.Н.	д.э.н., профессор, Ростовский государственный экономический университет (РИНХ)
Стредльцова Е.Д.	д.э.н., профессор, Южно-Российский государственный технический университет (НПИ)

Содержание

Введение

Управление предприятием или организацией предполагает принятие решений в условиях действия большого числа внешних факторов, наличия множества взаимодействующих элементов управляемой системы и ориентировано на достижение комплекса различных целей. Средством обеспечения эффективности принимаемых решений служит моделирование

Модель - искусственный объект, представляющий собой отображение (образ) системы и ее компонентов. Считается, что М моделирует А, если М отвечает на вопросы относительно А. Здесь М - модель, А - моделируемый объект (оригинал) [1].

Часто сложность управляемой системы и наличие значительной стохастической составляющей делают наиболее эффективным использование имитационных моделей. Имитационная модель обладает высокой степенью изоморфизма изучаемой системе, позволяет рассматривать значительное число деталей деятельности предприятия, учитывает случайные факторы. Эксперименты над имитационной моделью способствуют получению оценки различных вариантов предлагаемых управленческих решений.

Сегодня для построения имитационных моделей используют развитые системы, включающие графический интерфейс построения моделей. Наиболее распространенные средства: AnyLogic, Extend, Arena. Система Rockwell Arena – один из наиболее мощных и популярных инструментов имитационного моделирования – будет рассматривать в настоящем учебном пособии.

При изучении работы с Arena следует обратить внимание на книгу:

Simulation with Arena, 5th Edition by W. David Kelton, Randall P. Sadowski, Nancy B. Swets Rockwell Automation McGraw-Hill, 2010.

Это официальный учебник по имитационному моделированию в системе Arena, разработанный сотрудниками фирмы Rockwell Automation.

Вместе с системой Arena поставляется несколько электронных книг с документацией, из которых прежде всего вызывает интерес *Arena Users Guide*, где дается описание большинства модулей. Важным источником знаний являются и примеры моделей, представленные вместе с системой.

Кроме примеров моделей, с системой поставляются *SMARTS* – небольшие фрагменты моделей, отражающие тот или иной аспект использования Arena, например, работу с ресурсами или передачу транзактов между станциями.

На русском языке информации по системе существенно меньше. Большинство статей, материалов в сети Интернет и отдельные имеющиеся методические разработки посвящены описанию только модулей шаблона Basic Prrocess. Эти материалы можно использовать на начальном этапе изучения системы.

---

[1] Рекомендации по стандартизации. – М.: ГОССТАНДАРТ РОССИИ, 2001.

Что касается изучения методологии имитационного моделирования, то по ней имеется достаточно большой объем отечественной и переводной литературы. В частности, можно рекомендовать книгу:

Шеннон Р. Имитационное моделирование систем – искусство и наука. – М.: Мир, 1978. – 417 с.

Книга описывает основные идеи имитационного моделирования, цели и задачи построения моделей. В большинстве случаев именно по этой книге излагается в различных курсах и учебниках процесс имитационного моделирования.

Также следует обратить внимание на книгу:

Лоу А., Кельтон В. Имитационное моделирование. – СПб: Питер, 2004. – 848 с.

Эта книга раскрывает большинство аспектов имитационного моделирования: получение исходной информации, генерация случайных чисел, планирование имитационных экспериментов и т.д.

Что касается учебников по имитационному моделированию, то в большинстве случаев они ориентированы на какую-то конкретную систему имитационного моделирования. Значит, полезна для изучающего может быть только «общая часть» учебника, посвященная методологии моделирования. С другой стороны, преимуществом учебников является достаточно компактный объем и простота изложения. В частности можно рекомендовать книги:

Емельянов А. А., Власова Е. А., Дума Р. В. Имитационное моделирование экономических процессов. – М.: ФиС, 2009. – 416 с.

Томашевский В., Жданова Е. Имитационное моделирование в среде GPSS. – М.: Бестселлер, 2003. – 416 с.

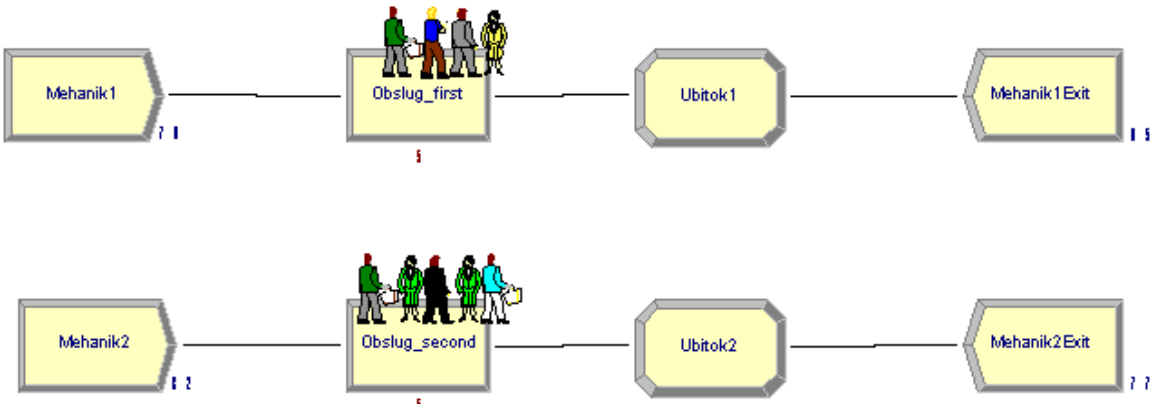
В настоящем учебном пособии будут раскрыты особенности использования системы Arena для построения имитационных моделей в экономике и управлении.

1 Основные понятия системы Arena

Имитационная модель в системе Arena представляет собой граф (рисунок 1), узлами которого являются **модули**. Модули связаны между собой с помощью соединений, по которым между модулями перемещаются **транзакты** [2].

Чтобы создать модель на языке Arena достаточно:

1) задать транзакты, ресурсы и другие объекты;
2) выбрать необходимые модули;
3) связать модули с помощью соединений;
4) задать параметры каждого из модулей;
5) задать характеристики модели в целом.



Модели в системе Arena являются процессно-ориентированными, граф модели показывает путь отдельного транзакта. Транзакт создается, ожидает в очереди, захватывает ресурсы, обрабатывается, освобождает ресурсы, уничтожается. Параллельно такой же путь могут проходить и другие транзакты, но задачу управления процессами берет на себя система.

Рассмотрим базовые понятия системы Arena, используемые при построении моделей. К таким понятиям следует отнести [3]:

- транзакт (entity);
- ресурс;
- атрибут;
- переменная;
- очередь (queue);
- расписание (schedule)
- блок
- модуль.

---

2 Транзакт – динамический объект имитационной модели.

[3] На самом деле есть и другие понятия – станция, последовательность. транспортер и т.д. Они будут рассмотрены далее в следующих главах.

Транзакт (entity, в терминах системы Arena) – это динамический объект имитационной модели, который перемещается между статическими узлами модели. Примеры транзактов – деталь, автомобиль, клиент, документ и т.д.

В модели может участвовать несколько транзакты разных типов, которые задаются разработчиком модели. Тип транзакта может меняться динамически, например, на каком-то этапе транзакт «машинокомплект» меняет свой тип на «автомобиль».

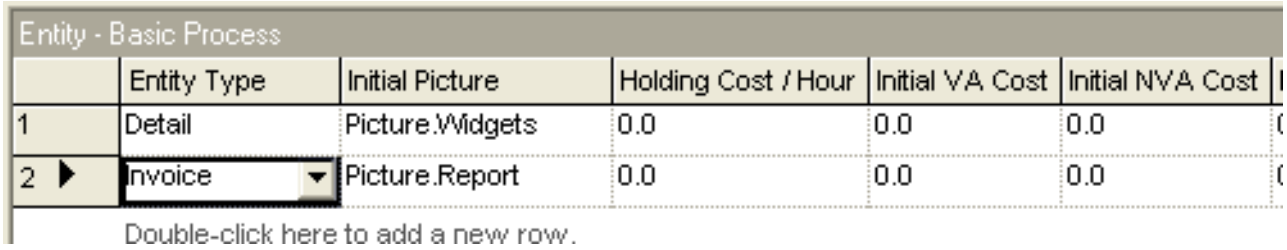
Транзакту соответствует картинка для анимации, с ее помощью можно следить за перемещениями транзактов между модулями и за обработкой транзактов.

Характеристики транзактов задаются с помощью **атрибутов**. Существуют системные атрибуты, установленные по умолчанию. Например, «время существования транзакта» (Total Time), «серийный номер транзакта» (Serial Number), тип транзакта (Entity Type) и т.д. Имеется атрибут «картинка» (Entity Picture), поменяв программно его значение можно изменить визуальное изображение транзакта.

Кроме того, пользователь может присваивать собственные атрибуты транзакта: «вес детали», «число покупок, совершенное покупателем супермаркета», «длина пакета данных», «сумма накладной» и т.д. Пользовательские атрибуты будут определять в дальнейшем время обработки транзакта, его приоритет при обработке, дальнейший путь транзакта.

Атрибуты транзактов позволяют собирать статистику по времени (сколько времени транзакт пробыл в модели, сколько обрабатывался, сколько ожидал и т.д.) и по стоимости (например, оценивать себестоимость изделия после прохождения технологического процесса).

Для создания транзактов используем невизуальный модуль Entity.

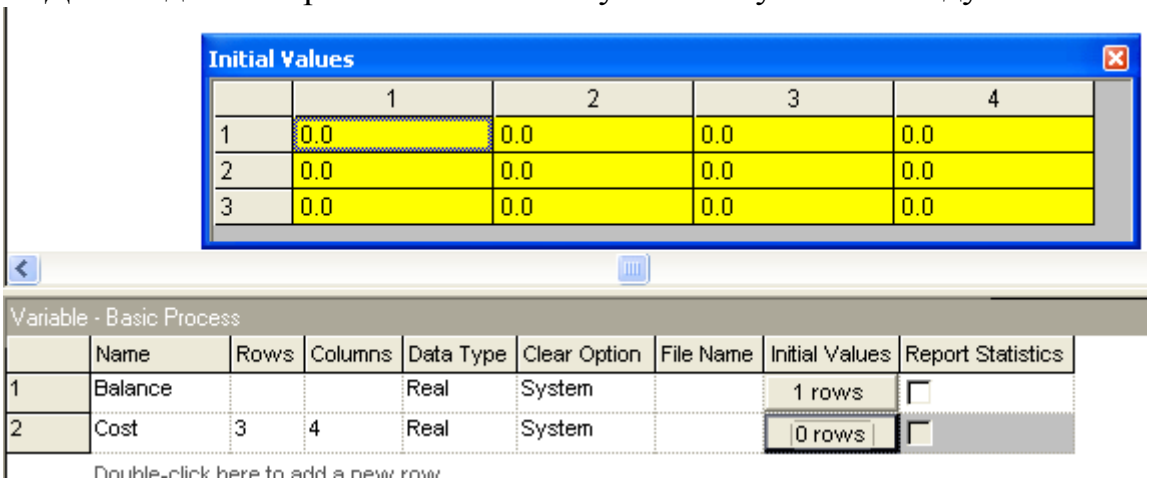


Переменная (Variable) – это глобальная переменная модели. Ее значение может считываться и изменяться во время моделирования. От значения переменной может зависеть, например, возможность прохождения некоторого участка: детали будут проходить обработку, если установлено значение некоторой глобальной переменной, свидетельствующей о том, что разрешение руководителя получено. Еще пример – счета могут проходить далее при наличии некоторой минимальной суммы в кассе предприятия. При оплате каждого счета это значение меняется [4].

[4] Важно отличать атрибут транзакта от переменной. Значение переменной одно на всю модель. В то же время у каждого транзакта может быть свое значение некоторого атрибута. В наших примерах – число денег в кассе – это переменная, глобальное единственное

Переменные могут также быть одномерными и двумерными массивами. Обращение к элементам массива осуществляется так же как в обычных языках программирования *A(5)* или *B(3,2)*.

Для создания переменных используем невизуальный модуль Variable.



Здесь задается имя переменной. Если это массив – нужно задать число строк Rows и столбцов Columns (для одномерных – только строк, Rows). Кнопкой в колонке Initial Values вводим начальное значение (или для массивов - значения) переменной. Это значение переменная будет иметь при старте прогона модели.

Существуют и системные переменные, важнейшая из них – *TNOW*, текущее значение модельного времени [5].

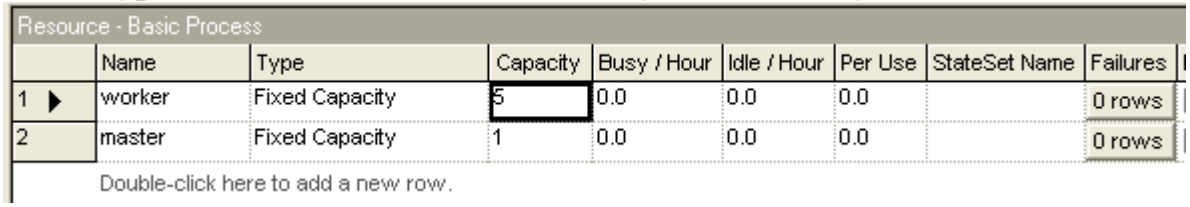
Ресурс. Ресурсы разделяются между транзактами, транзакты конкурируют за ресурсы. Например, детали обрабатываются на станке, клерк рассматривает заявления, сервер обрабатывает запросы и т.д.

По ходу моделирования транзакт захватывает ресурс (если не может захватить – ожидает пока ресурс освободится), использует его перемещаясь по модели, затем отпускает ресурс.

Важнейшая характеристика ресурса – *мощность*, число единиц ресурса. Примеры: число каналов для телефонных разговоров, число окошек в банке, число одинаковых станков и т.д.

Arena позволяет задать для ресурса характеристики стоимости – сколько стоит час единицы ресурса когда он свободен, занят, сколько стоит одно обращение к ресурсу и т.д.

Ресурсы задаются с помощью невизуального модуля resource.

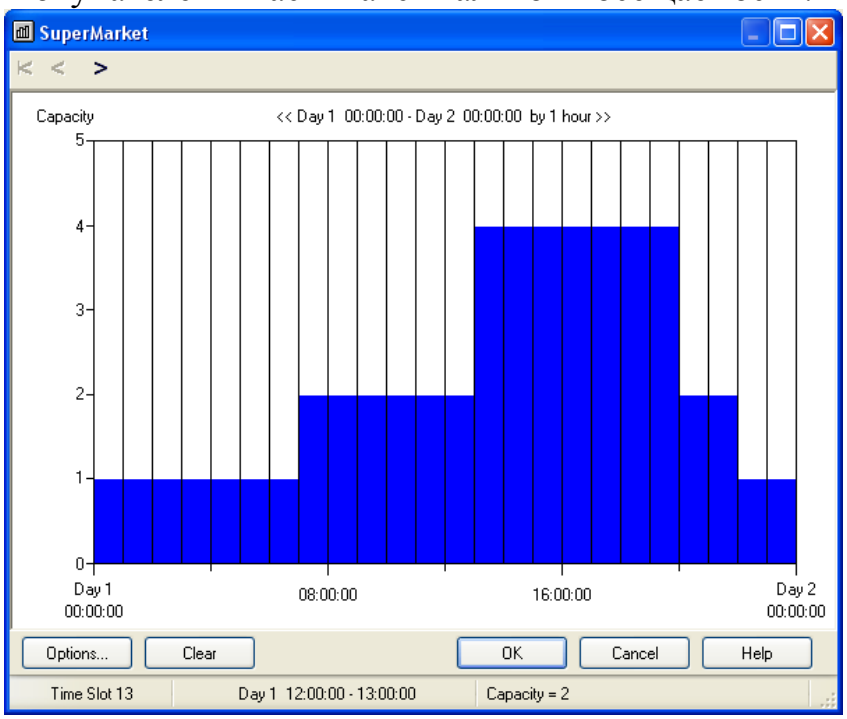


значение,. А вот сумма документа «Счет-фактура» - это атрибут. У каждой счет-фактуры своя сумма.

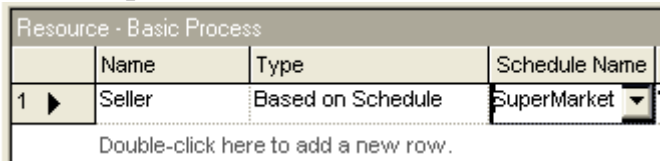
[5] В базовых единицах времени. Базовые единицы выставляются в пункте меню Run – Setup вкладка Replication parameters.

Здесь для обоих ресурсов задано имя и указано, что ресурс имеет фиксированную мощность. И задана мощность (capacity) – в цеху работает один мастер и пятеро рабочих.

Для ресурса можно задать расписание. Например, продавцы супермаркета могут приходить на работу по определенному графику с целью лучше обслуживать покупателей в часы максимальной посещаемости.



Расписание задается в невизуальном модуле Schedule. При создании ресурса с расписанием вместо опции «Fixed capacity» указываем «by Schedule» и выбираем это расписание.



В ходе моделирования ресурс принимает одно из следующих *состояний* [6]:

- IDLE (свободен);
- BUSY (занят);
- FAILED (неисправен);
- INACTIVE (отключен).

Пользователь может создавать и собственные состояния.

Очередь (**queue**) – может образовываться возле тех модулей, которые мешают транзакту пройти дальше. Пример – станок для обработки деталей занят, и деталь ожидает в очереди, пока он освободится. Шлагбаум на железнодорожном переезде закрыт (идет поезд) и поэтому автомобили образуют очередь, чтобы ожидать открытия шлагбаума. Еще пример – детали нужно

[6] Для ресурсов с мощностью большей единицы состояние следует трактовать так: IDLE – все единицы свободны, BUSY – хотя бы одна занята.

упаковать по 12 штук. По мере поступления деталей они будут ожидать упаковки в очереди, до тех пор, пока не появятся 12-ая.

Очереди создаются автоматически, но разработчик модели имеет возможность управлять их свойствами. Важнейшим свойством очереди является ее тип, определяющий поведение транзактов в очереди. Существуют четыре типа:

- *FIFO – «First in first out»*. Первый пришел - первый вышел. Наиболее распространенный случай, используется по умолчанию;

- *LIFO – «Last in first out»*. Последним пришел - первым вышел. Такая очередь называется «стек». Пример – партии товара помещаются в хранилище. По мере освобождения упаковочной машины товар достается, упаковывается и передается дальше. Склад устроен так, что поступившие вновь товары закрывают доступ к поступившим ранее (вспомним междугородние автобус и его багажное отделение);

- *Lowest Attribute Value.* По наименьшему значению атрибута;

- *Highest Attribute Value.* По наибольше значению атрибута.

Последние два метода позволяют реализовать очередь с *приоритетом*. Для этого нужно завести искусственный пользовательский атрибут, указывающий приоритет для каждого типа транзакта. Например, машины скорой помощи (значение атрибута «1») будут попускать вперед. Если в очереди три обычных машины (значение атрибута «0»), то прибывшая машина скорой помощи займет место перед ними – т.е. в начале очереди. А вот следующая машина скорой помощи займет место после первой машины скорой помощи перед обычными машинами.

Приоритет может базироваться и на каком-то естественном, содержательном атрибуте, например, вес детали, сумма денежного перевода и т.д.

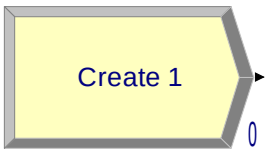
Контрольные вопросы

1) В чем заключается разница между атрибутом и переменной?
2) Каким образом можно организовать очередь с приоритетом?
3) Что такое тразнакт?
4) Что означает характеристика ресурса «Capacity»?
5) Что означают состояния ресурса «BUZY» и «IDLE»?
6) В каких случаях модуль будет иметь возле себя очередь?
7) Можно ли менять значение переменной во время моделирования?
8) Сколько значений атрибутов имеется в модели во время моделирования?
9) Библиотекарь обслуживает подходящих к нему читателей. Чем будут «библиотекарь» и «читатель» с точки зрения системы Arena?
10) Каждый читатель хочет взять какое-то количество книг. Чем будет «количество книг, которые хочет получить читатель» с точки зрения системы Arena?
11) Библиотека платная и за каждую взятую книгу читатель должен внести 10 рублей. Чем будет «полученная в течение дня выручка библиотеки» с точки зрения системы Arena?
12) Директор магазина выставляет большее число кассиров, чтобы облегчить обслуживание покупателей в «час пик». Как это отразить в системе Arena?

2 Базовые модули системы Arena

Модуль **Create (генератор транзактов) –** вводит транзакты в модель. Например, покупатели приходят в магазин, суда прибывают в порт, в службу поддержки поступают звонки и т.д.

Обозначение:



Модель может включать несколько модулей Create.

На рисунке показано окно параметров для модуля Create, где можно задать, например, как часто должны поступать транзакты.

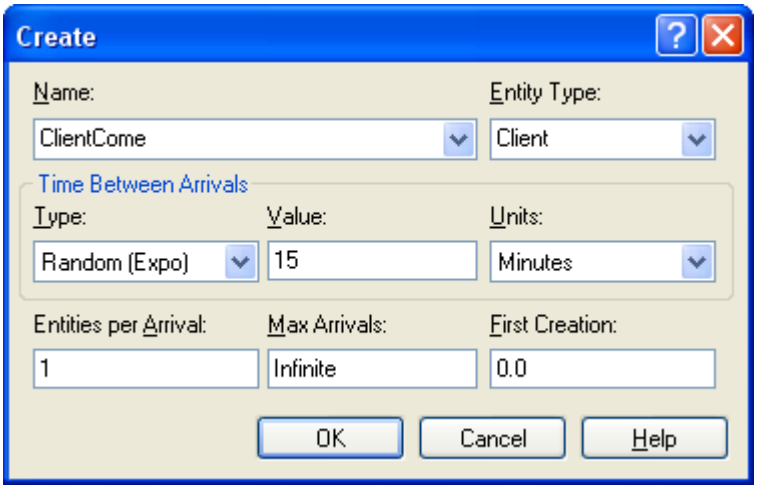


Рисунок говорит о следующем: клиенты парикмахерской заходят примерно раз в 15 минут, закон распределения интервала между клиентами – экспоненциальный.

Для модуля задаются такие параметры:

- название модуля (Name [7]);
- тип поступающего транзакта (Entity type);
- интервал между прибытиями транзакта (Time Interval);
- число транзактов при каждом поступлении (Entities per arrival);
- максимальное число поступлений (Max Arrivals).

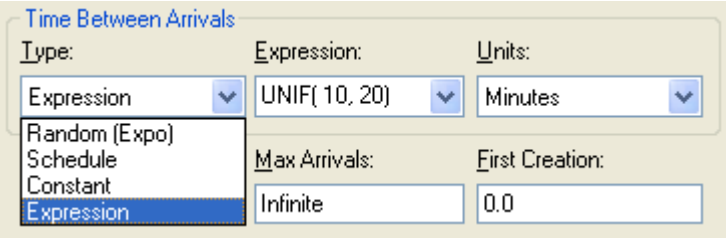
Интервал может быть задан как:

- «константа» (Constant). Транзакты поступают в модель строго через равные промежутки времени. Например, деталь поступает на участок обработки строго один раз в минуту

[7] Настойчиво рекомендуем сразу же при создании присваивать всем модулям содержательные, «говорящие» имена. В противном случае понимание модели будет затруднено.

- «по расписанию» (Schedule). В этом случае следует выбрать расписание. Например, в течение дня у магазина наблюдается разная интенсивность поступления клиентов – в «часы пик» она значительно увеличивается.

- «выражение» (Expression) Указывается функция распределения и ее параметры. Так, на рисунке показано, что интервал поступления транзактов подчиняется равномерному закону распределения и может составлять от 10 до 20 минут *unif(10,20).*



- «случайный» (Random(Expo)). В системах массового обслуживания очень часто встречается пуассоновский поток заявок. При таком потоке интервал времени между появлениями заявок подчиняется экспоненциальному закону распределения. Для отражения этого частого случая и предусмотрена опция «Random (expo)». Можно выбрать ее и задать математическое ожидания интервала. Эта опция введена для удобства. С тем же результатом можно использовать выражение *expo(15).*

Задавая интервал необходимо выбрать единицу измерения времени [8].

Для модуля Create задается число транзактов при каждом поступлении (Entities per arrival). Чаще всего поступает один транзакт, но модуль Create может вводить транзакты партиями. Например, согласно технологическому процессу, в цех поступают одновременно 5 деталей (рабочий раз в 7-10 минут привозит тележку, с пятью деталями). Мы можем считать детали транзактами и указать, что они всегда поступают по 5.

Можно смоделировать и ситуацию, когда люди заходят в магазин группами 1-4 человека. В поле «Entities per arrival» указываем выражение *unif(1,4),* значит при каждом поступлении прибывает от одного до четырех транзактов.

В поле «максимальное число поступлений» (Max Arrivals) обычно стоит значение по умолчанию «Infinite», бесконечность. Поле позволяет ограничить число транзактов в модели. Например, грузовики прибывают в гараж, где их проверяют, обслуживают и ставят на ночь. Но у фирмы ограниченное число грузовиков и, значит, в какой-то момент поступит последний.

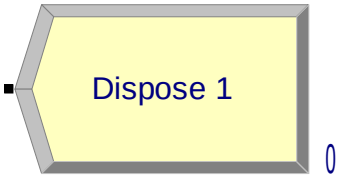
Как правило, эту возможность используют для так называемых, «замкнутых схем». «Открытая схема» – транзакты приходят в модель (генерируются, создаются), затем после необходимого обслуживания, уничтожаются, выводятся из модели. При «закрытой схеме» все выглядит иначе – в модель

[8] Небрежность при выборе единиц измерения времени – одна из наиболее распространенных ошибок при создании моделей.

вводится определенное число транзактов, которые затем циркулируют внутри модели. Например, у фирмы 10 автобусов. Они проходят обслуживание и уходят на маршрут, затем возвращаются и вновь уходят и т.д.

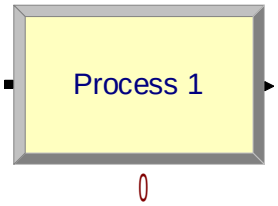
Модуль **Dispose (терминатор транзактов) –** выводит транзакты из модели (уничтожает транзакты). Например, покупатели после покупки товара, покидают магазин.

Обозначение:



Модуль **Process** (обработка, действие) **–** моделирует обработку транзактов: обслуживание покупателей продавцом, разгрузку судна, ремонт телевизора и т.д. Обратим внимание, что с точки зрения имитационной модели «обработка» означает задержку транзакта на определенное время, содержательный смысл этой задержки определяется исследователем.

Обозначение:



Обработка транзактов может потребовать использования ресурсов. Так, для сверления детали нужен соответствующий станок, для обслуживания покупателя – продавец и т.п.

Арена обеспечивает достаточную гибкость работы с ресурсами. Обработка может потребовать нескольких единиц ресурсов, либо может потребовать несколько разных ресурсов. Например, для профилактического осмотра автомобиля нужны два рабочих автосервиса и смотровая яма. Транзакт может «захватить» ресурс и перемещать его за собой по модели, с тем, чтобы потом «отпустить». Автомобиль «захватывает» яму, затем его осматривает одна бригада рабочих, затем другая бригада выполняет кузовные работы, затем автомобиль «отпустит» яму.

Существует четыре типа модуля Process:

- *Delay*, задержать. Не требуется никаких ресурсов. Пример: покрашенный стол выставляется на улицу, где он должен просохнуть. Столы не конкурируют ни за какой ресурс – одновременно на солнце может сушиться неограниченное число столов.

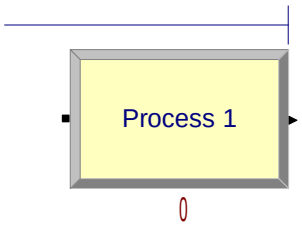
- *Seize-Delay-Release*, захватить-задержать-отпустить. Пример: деталь поступила на станок, была просверлена и освободила станок. Это наиболее распространенный вариант модуля.

-*Seize-Delay*, захватить – задержать.

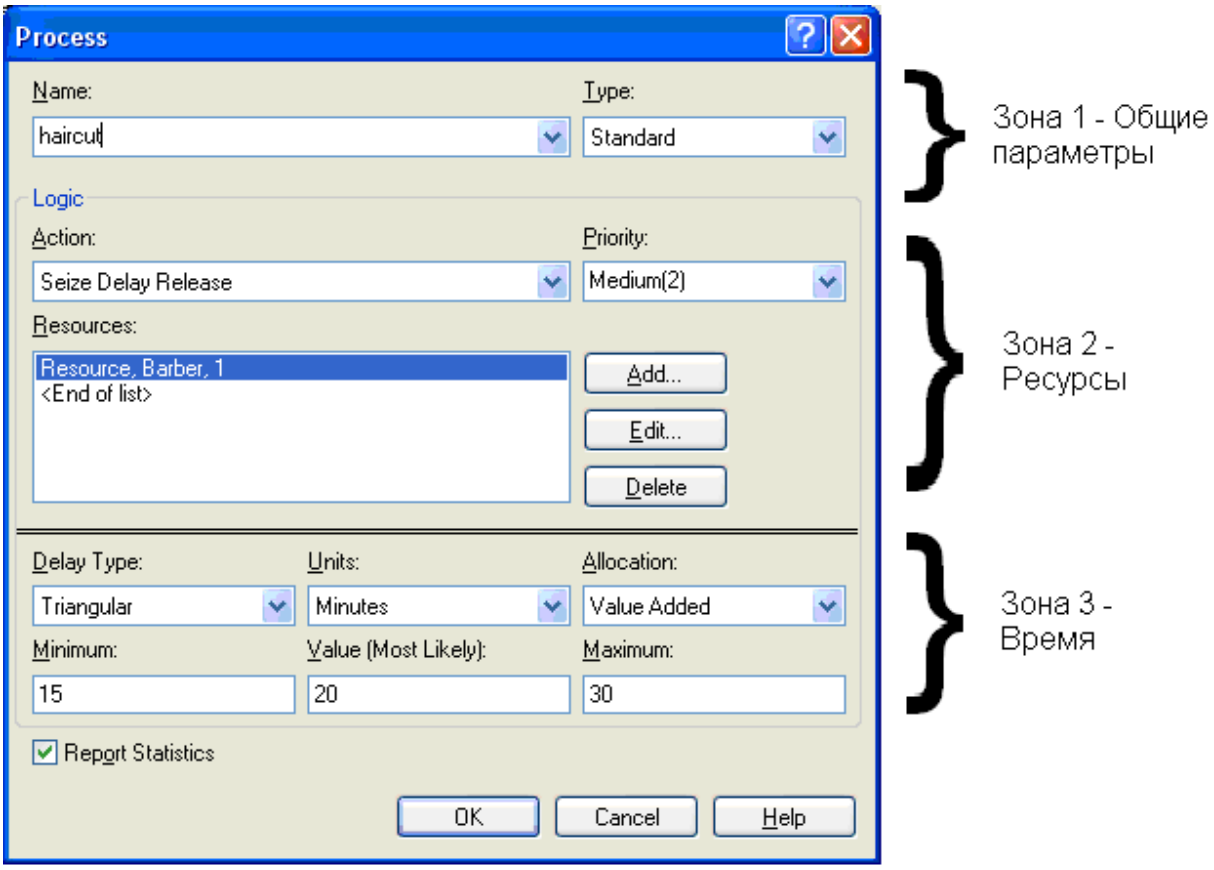
- *Delay-Release*, задержать-отпустить.

Два последних варианта используются для построения сложных многофазных процессов, предполагающих несколько стадий обработки. Захватив ресурс, транзакт может удерживать его несколько операций.

В случае, если модуль Process предполагает захват какого-то ресурса (то есть типы Seize-Delay-Release или Seize-Delay), к модулю автоматически добавляется очередь. Если нужного ресурса нет в наличии (сломан, неактивен согласно расписанию или занят другими транзактами) транзакт будет ожидать его освобождения в очереди:

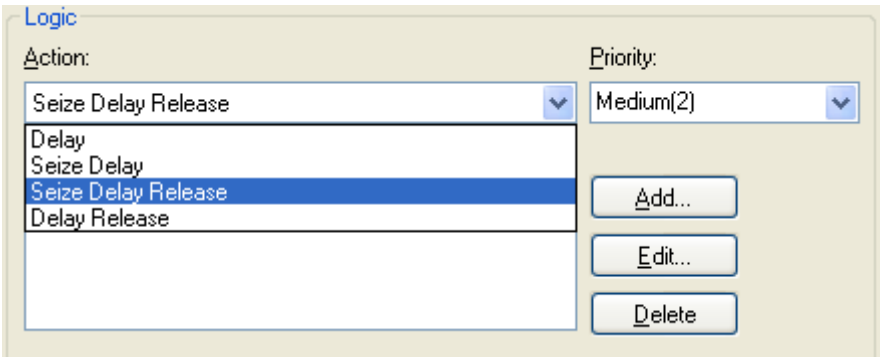


Пример модуля Process приведен на рисунке. Как видим, здесь используется тип Seize-Delay-Release. Процесс носит имя «стрижка». Для процесса необходим один парикмахер. Время стрижки распределено по треугольному закону распределения, причем на стрижку может уйти от 15 минут (минимальное значение) до 30 минут (максимальное) при том, что 20 минут – это наиболее вероятное значение.



Рассмотрим более подробно параметры модуля. Окно распадается на три зоны. В первой зоне заполняем имя операции - Name (так же как для любого другого модуля Арены).

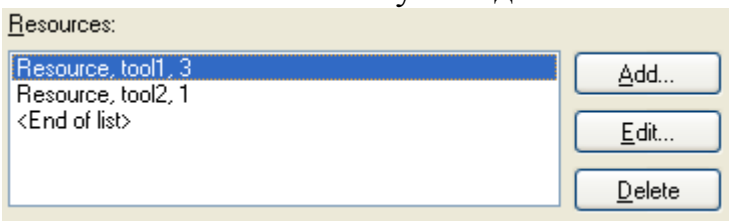
Вторая зона отвечает за тип процесса и за используемые ресурсы.



В списке Action нужно выбрать необходимый тип модуля.

Если выбран тип Delay, то больше ничего указывать не нужно, никакие ресурсы для операции не требуются.

Если же выбран другой тип, то нужно явно указать, какие именно ресурсы и в каких количествах нужны для выполнения операции:

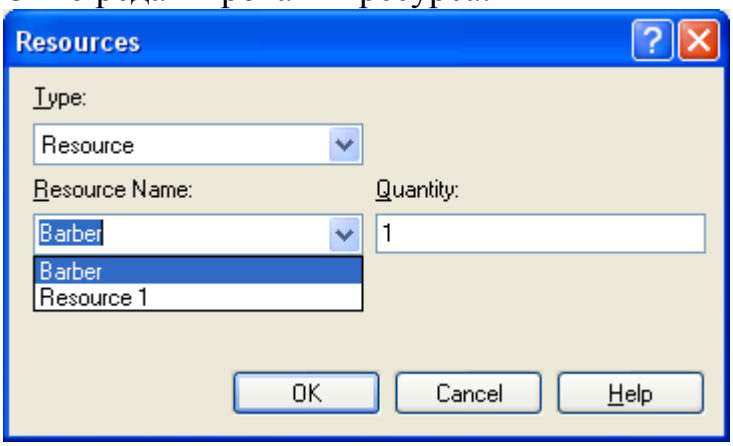


На рисунке выше показано, что для выполнения операции нужны три штуки первого инструмента и одна штука второго инструмента.

Операция может потребовать нескольких единиц ресурсов, например, для разгрузки большого судна нужно два портовых крана.

С помощью кнопок Add (добавить), Edit (редактировать), Delete (удалить) можно управлять списком необходимых ресурсов.

Окно редактирования ресурса:

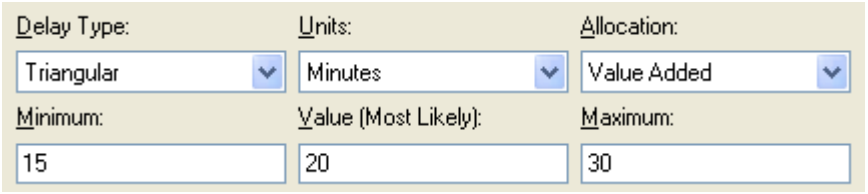


В окне следует выбрать из списка имя ресурса (предполагается, что ресурсы мы создали заранее. Можно, однако, ввести имя несуществующего пока ресурса, в этом случае ресурс будет создан автоматически). В поле Quantity (количество) вводим число единиц ресурса, необходимых для выполнения операции над одним транзактом [9].

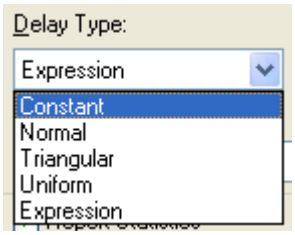
В окне редактирования ресурсов имеется еще одно поле – Type. Здесь можно выбрать один из двух вариантов: Resource (конкретный ресурс) или Set (множество, пул ресурсов). Работа с пулами ресурсов будет рассмотрена в главе 6.

В поле Priority нужно указать приоритет операции. Дело в том, что один и тот же ресурс может использоваться в нескольких модулях Process. С помощью поля приоритета можно указывать, какая из операций, требующая этого ресурса начнется первой [10]. Например, сотрудник паспортного стола выполняет две операции – «прием заявления» и «выдача паспорта». Приоритет второй операции выше – это значит, что при наличии людей, ожидающих обоих операций сотрудник в первую очередь станет обслуживать тех, кто пришел получить готовый паспорт.

Вне зависимости от выбранного типа модуля Process необходимо задать время исполнения операции (зона 3):



В целом, работа ведется аналогично модулю Create. Нужно указать способ определения времени, единицу измерения времени. Разработчики системы по умолчанию предлагают треугольный закон распределения с тремя параметрами – минимальным, наиболее вероятным и максимальным временем исполнения операции.



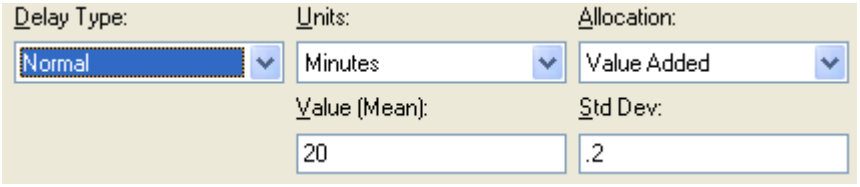
[9] Не путаем это значение с мощностью ресурса, то есть числом единиц, которое имеется в модели! Например, в мастерской может работать пять рабочих, а для ремонта кузова одного автомобиля нужна совместная работа двух рабочих.

[10] Необходимо понимать разницу между ресурсом, например, станком и процессом, например обработкой детали на станке. Модуль Process представляет именно операцию, протекающий во времени процесс, стадию обработки. Эта разница должна отражаться и в названиях, которые присваиваются модулю Process и ресурсам.

Другие варианты:

- фиксированное время (Constant), операция всегда занимает четко заданное время.

- нормальное распределение времени обслуживания (Normal). В этом варианте следует задать два параметра – математическое ожидание и среднеквадратическое отклонение (как показано на рисунке).



- равномерное распределение времени обслуживания (Uniform). Задается минимальное и максимальное время.

- выражение (Expression). Можно ввести выражение, показывающее необходимое время обслуживания. Например, если время обслуживание подчиняется экспоненциальному закону, то можно указать *expo(15)*.

Выражения позволяют более гибко управлять временем обработки транзактов. Например, транзакт-вагон может иметь атрибут *mass*, показывающий вес груза. А время разгрузки может зависеть от веса: *10*mass*. Чтобы включить элемент случайности *expo (10*mass)*.

Последнее поле Allocation позволяет отнести время, проведения операции к одной из категорий, для последующего анализа. Например, в данном случае указано, что операция относится к «создающим стоимость» (Value Added). В дальнейшем по результатам моделирования мы сможем определить какой процент времени тратится на создание стоимости, какой – на вспомогательные операции, какой – на ожидание. Так можно судить об эффективности моделируемых процессов.

После того, как мы изучили три первых модуля [11], можем приступить к созданию первой простейшей модели.

Пример 1. Клиенты заходят в парикмахерскую раз в 15 минут (закон распределения интервала времени прихода клиентов экспоненциальный). В парикмахерской работают два парикмахера, которые тратят на стрижку клиента от 20 до 40 минут, закон распределения – равномерный.

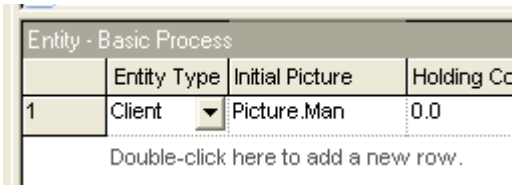
Прежде всего, необходимо определиться с важнейшими понятиями:

- клиент – это транзакт.

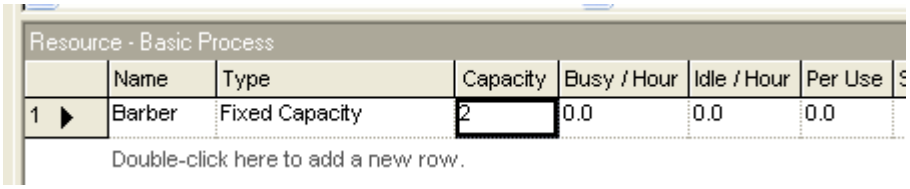
- парикмахер – это ресурс.

[11] Модули, которые мы рассмотрели, относятся к шаблону Basic Processes. Следующий шаблон Advanced Process содержит модули, позволяющие строить более сложные и гибкие модели. К этому шаблону относятся и модули Sieze, Delay, Release. Их функционал почти полностью сходен с функционалом модуля Process. Кроме того, модули Sieze позволяет использовать общие очереди к одним и тем же ресурсам

Создадим с помощью невизуального модуля Entity (шаблон Basic Process) транзакты-клиенты. Дадим транзакту имя и выберем подходящую картинку:



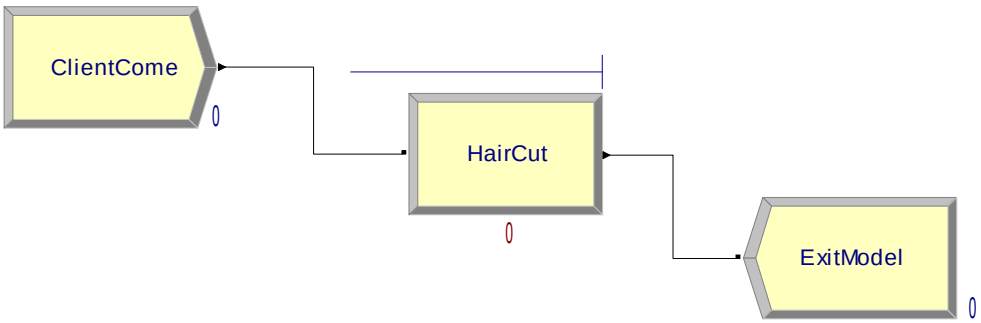
Теперь с помощью невизуального модуля Resource создадим парикмахера. Здесь мы задали имя ресурса – Barber:



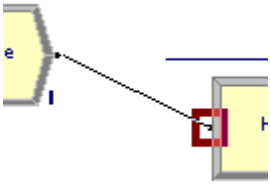
В поле Type укажем, что используется фиксированная мощность ресурса (fixed capacity). Далее в поле capacity (мощность) задаем число единиц ресурсов, имеющихся в наличии – 2 парикмахера.

Остальные параметры мы не используем в модели, эти параметры отвечают за стоимость ресурсов: Busy/Hour (стоимость часа ресурса если он занят), Idle / Hour (стоимость часа ресурса, если он простаивает), Per Use (стоимость единичного обращения к ресурсу).

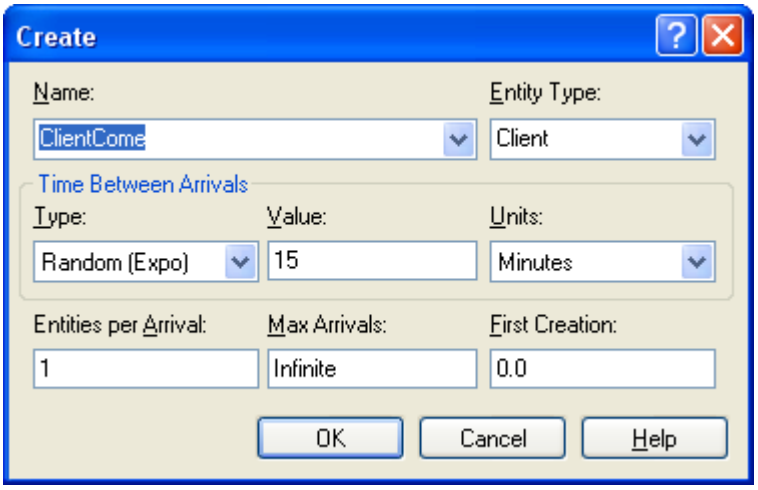
Теперь можно создавать граф модели:



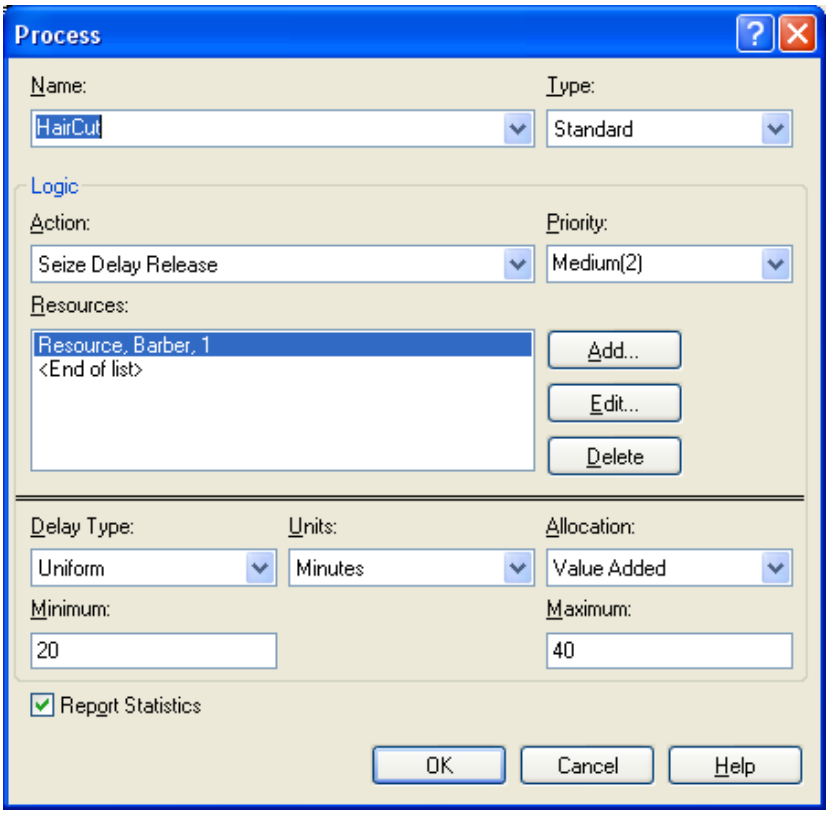
Чтобы нарисовать граф достаточно вытянуть на рабочую плоскость необходимые модули один за другим, при этом они будут соединяться автоматически. Также можно использовать инструмент соединения с панели инструментов  .



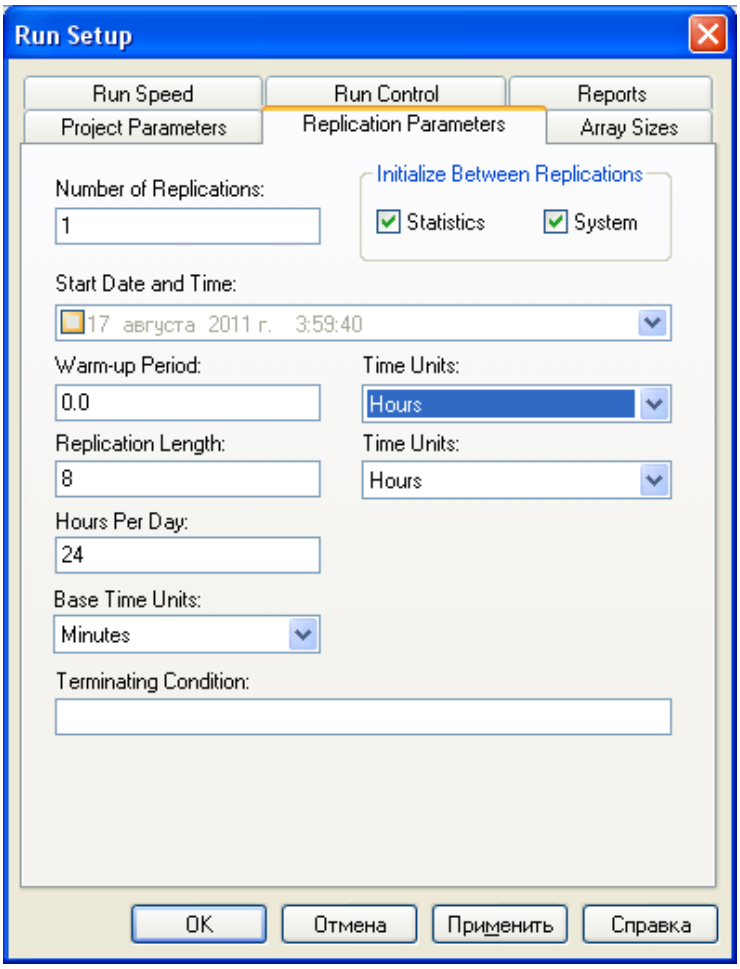
Рассмотрим параметры отдельных модулей. На рисунке показано окно параметров модуля Create:



На следующем рисунке приведено окно параметров модуля Process. У модуля Process имеется очередь. Обратим внимание, что это единая очередь для всех мастеров, как только освобождается один из мастеров, первый клиент в очереди садится в его кресло.



Теперь необходимо определить параметры модели в целом. Для этого обращаемся к пункту главного меню Run-Setup. Здесь нас интересует вкладка Replication Parameters.



Здесь можно задать число повторений прогона модели «Number of Replication».

Далее определяем длину интервала моделирования «Replication Length», в примере мы уставили 8 часов. Если оставить значение по умолчанию – работа будет продолжать бесконечно до того, как пользователь остановит моделирование или пока наступит условие окончания.

В окне можно указать так называемый Warm-up период, результаты которого учитываться не будут [12].

Наконец, очень важно выбрать в Base Time Units единицу времени, наиболее подходящую для конкретной модели. Именно в таких единицах будут выводиться отчеты о моделировании, а также именно она будет использоваться в специальных функциях, которые обращаются к времени.

Поле Terminating Condition позволяет указать условие завершения моделирования. Например, мы могли бы задать здесь выражение, которое завершит моделирование при успешном обслуживании 100 клиентов.

[12] Назначение периода warm up будет рассмотрено позже в главе 9.

Посмотрим на приведенный ниже листинг, он получен с помощью пункта меню Run – SIMAN - VIEW:

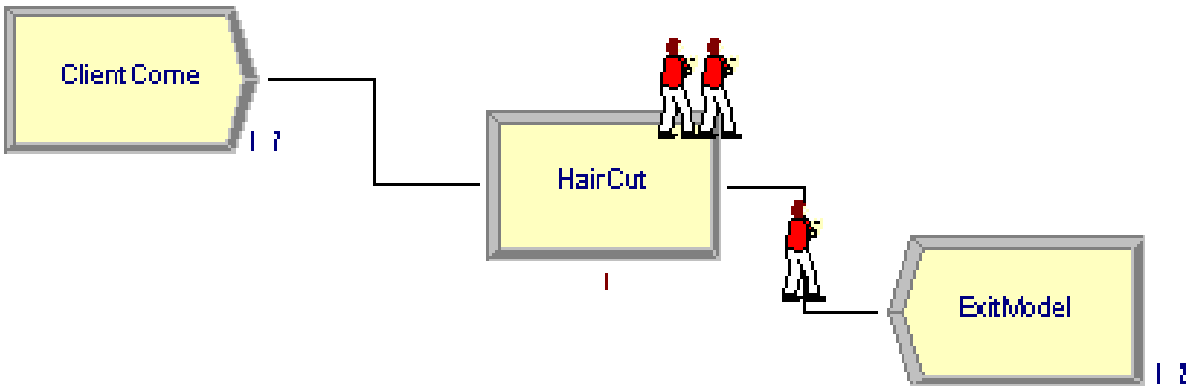
```
;     Model statements for module:  BasicProcess.Create 1
(ClientCome)
;
2$            CREATE,
1,HoursToBaseTime(0.0),Client:HoursToBaseTime(EXPO(15)):NEXT(3$)
;
3$            ASSIGN:        Client-
Come.NumberOut=ClientCome.NumberOut + 1:NEXT(1$);
;
;     Model statements for module:  BasicProcess.Process 1
(HairCut)
;
1$   ASSIGN:        HairCut.NumberIn=HairCut.NumberIn + 1:
                    HairCut.WIP=HairCut.WIP+1;
9$   QUEUE,         HairCut.Queue;
8$   SEIZE,         2,VA:
                    Barber,1:NEXT(7$);
7$   DELAY:         Uniform(20,40),,VA;
6$   RELEASE:       Barber,1;
54$  ASSIGN:        HairCut.NumberOut=HairCut.NumberOut + 1:
                              HairCut.WIP=HairCut.WIP-1:NEXT(0$);
;     Model statements for module:  BasicProcess.Dispose 1 (Ex-
itModel)
0$            ASSIGN:        ExitMo-
del.NumberOut=ExitModel.NumberOut + 1;
     57$           DISPOSE:       Yes;
```

Этот программный код был автоматически сформирован системой Arena на основе созданного нами графа и его параметров. В тексте программы легко угадываются работа отдельных модулей. Мы видим, что для каждого из модулей потребовалось несколько строк программного кода.

Теперь запустим модель с помощью специальной панели инструментов:

Эта панель позволяет запускать приостанавливать модель, прокручивать ее без анимации, регулировать скорость моделирования, выполнять моделирование по шагам.

Вид графа модели во время моделирования показан на рисунке. Видны перемещающиеся между модулями транзакты.



На рисунке можно увидеть следующее: поступили 17 клиентов; два клиента находятся в очереди на стрижку; один парикмахер занят; только что освободился второй парикмахер – его клиент покидает модель.

На самом деле с точки зрения модельного времени изменения происходят мгновенно, время на перемещение между блоками не тратится. Как только обслуживание клиента закончилось, он покидает модель, а следующий занимает его место. Арена демонстрирует перемещение транзактов между модулями для наглядности.

Рассмотрим теперь модель с несколькими процессами.

Пример 2. Имеется некоторый технологический процесс. Поступающие детали (интервал поступления 10 минут, закон распределения экспоненциальный) помещаются на специальную тележку (имеется 3 таких тележки) и подвергаются грунтовке (операция выполняется 8±2 минут рабочим №1), затем окраске (6±2 минуты, рабочим №2), затем проходят просушку в течение 20 минут, затем снимаются с тележки и освобождают цех.

Очевидно, что для данного примера:
- транзакт – деталь;
- ресурсы: тележка, рабочий №1, рабочий №2.

Окно создания ресурсов:

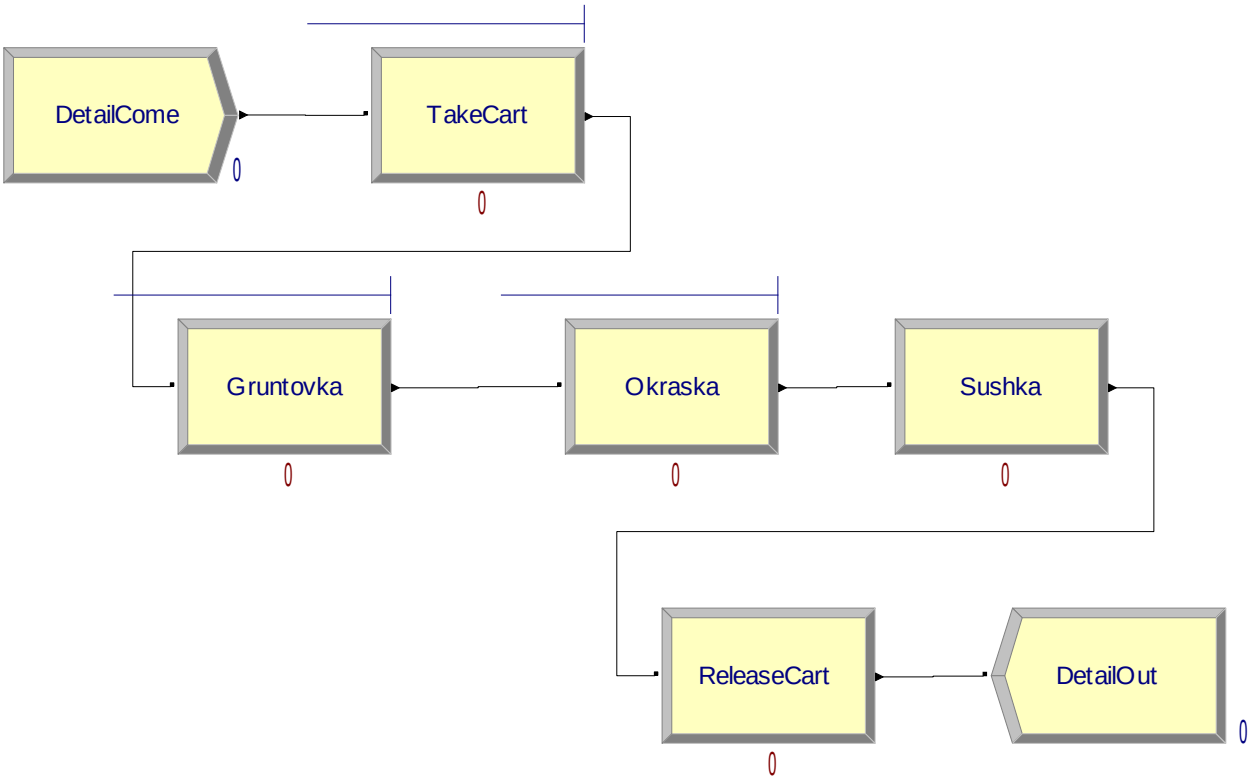
Resource - Basic Process

	Name	Type	Capacity
1	cart	Fixed Capacity	3
2	worker1	Fixed Capacity	1
3	worker2	Fixed Capacity	1

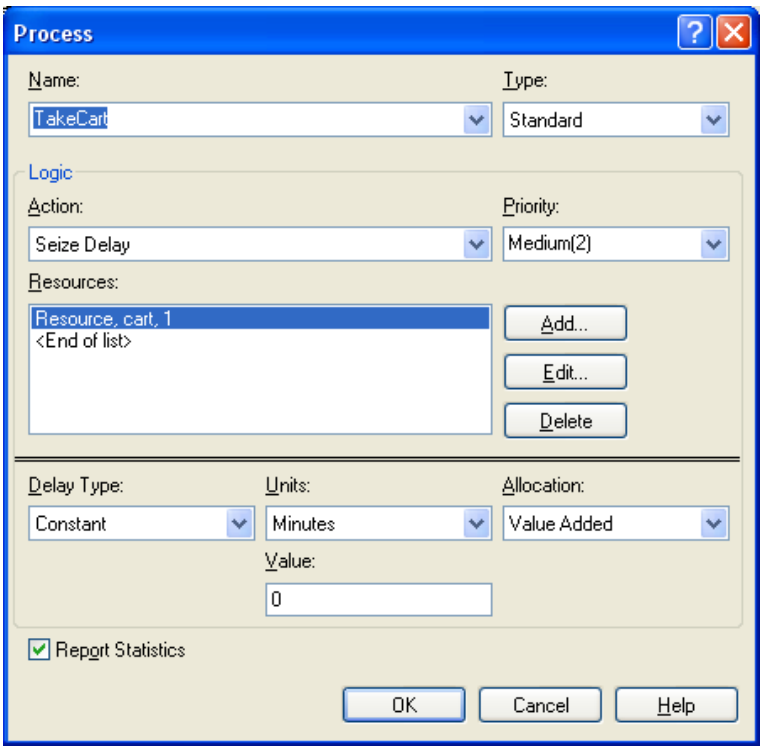
Double-click here to add a new row.

На следующем рисунке показан граф модели. Обратим внимание на одну особенность - ресурс «тележка» захватывается транзактом в модуле Take Cart и удерживается в течение нескольких операций.

Рассмотрим модель и ее составляющие более подробно. Транзакты-детали поступают в модель с помощью модуля DetailCome типа Create. Далее происходит захват тележки (cart) в модуле TakeCart.

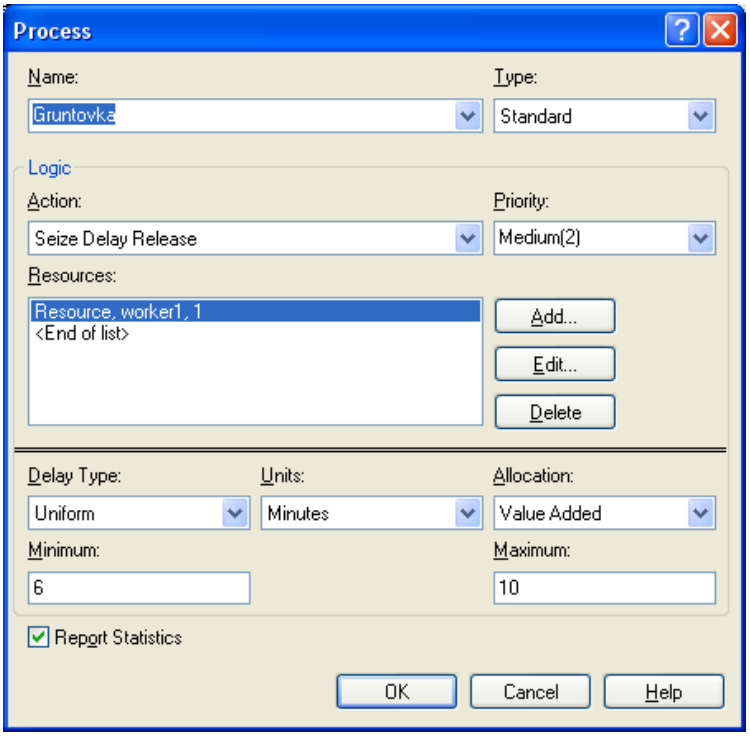


Как видим из следующего рисунка, выбран тип модуля Seize-Delay. После прохождения модуля деталь будет удерживать за собой ресурс-тележку. Время задержки в модуле нулевое.



Далее осуществляется грунтовка и окраска. Это модули типа Seize-Delay-Release, ресурсом выступают рабочие (worker1 для грунтовки, worker 2 для окраски). Время выполнения операции подчинено равномерному закону.

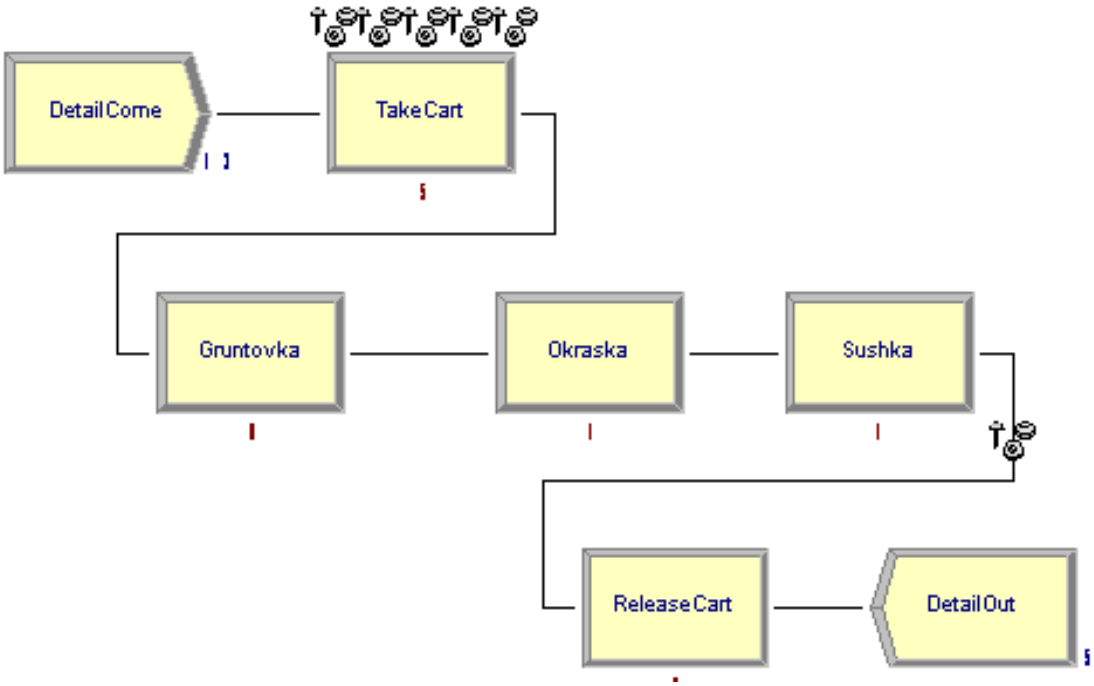
Так для грунтовки выставлено минимальное время 6, максимальное 10 минут, в соответствии с условиями задачи:



Модуль Process, соответствующий операции сушки детали имеет тип Delay, это значит, что никаких дополнительных ресурсов (кроме тележки, которую по-прежнему удерживается деталью) для операции не нужно. Время отводимое на сушку – строго 20 минут, задано с помощью константы.

Обратим внимание, что во всех модулях, предполагающих захват ресурса появляются очереди. В них детали будут оставаться в случае, если не окажется свободного ресурса до тех пор, пока такой ресурс не освободится.

Граф модели в процессе прогона:



Фрагмент отчета по результатам моделирования:

Waiting Time

	Average
Gruntovka.Queue	1.1864
Okraska.Queue	0.08008916
TakeCart.Queue	31.4936

Instantaneous Utilization

	Average
cart	0.9344
worker1	0.6700
worker2	0.4797

По результатам моделирования видно, что «узким местом» является модуль Process «TakeCart», то есть детали ожидают тележки. Это видно из рисунка, а также из фрагментов отчета: разница времени нахождения в очередях и разница в загрузке ресурсов.

Для повышения эффективности системы добавим еще одну тележку (изменим мощность, capacity ресурса «тележка» в невизуальном модуле Resources).

Отчет после прогона измененной модели:

Waiting Time

	Average
Gruntovka.Queue	2.4048
Okraska.Queue	0.03015598
TakeCart.Queue	12.9197

Instantaneous Utilization

	Average
cart	0.8331
worker1	0.7699
worker2	0.5623

Время ожидание в очереди значительно сократилось, загрузка ресурсов выровнялась, производительность системы возросла.

Контрольные вопросы

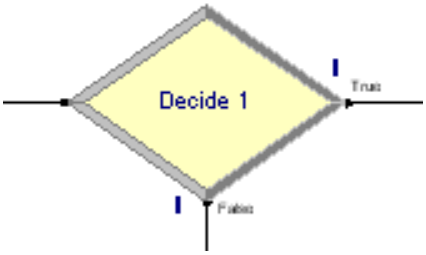
1) Какой модуль отвечает за обслуживание транзакта?
2) В каких случаях модуль Process будет иметь очередь?
3) Какой типа модуля Process следует выбрать для операции, соответствующей переходу транзакта-работника к месту выполнения работы?
4) Может ли транзакт в модуле Process использовать сразу несколько единиц ресурса?
5) Может ли транзакт в модуле Process использовать сразу несколько различных ресурсов?
6) Каким образом нужно настроить модуль Create чтобы организовать «замкнутую схему», при которой ограниченное число транзактов будет циркулировать внутри модели не покидая ее?

7) Может ли интервал времени между приходом транзактов быть случайной величиной?
8) Какова важнейшая характеристика ресурса, которую можно получить в результате моделирования?
9) Что делает модуль Dispose?
10) Как сделать, чтобы в отчетах по результатам моделирования время выводилось в минутах?
11) Если транзакт «захватил» ресурс и удерживает его, перемещаясь по модели, как он сможет «отпустить» ресурс?
12) Как задать графическое изображение транзакта?

3 Ветвление. Использование выражений

Модуль ветвления **Decide** – направляет входящие транзакты по одной из исходящих ветвей в зависимости от условия или случайным образом.

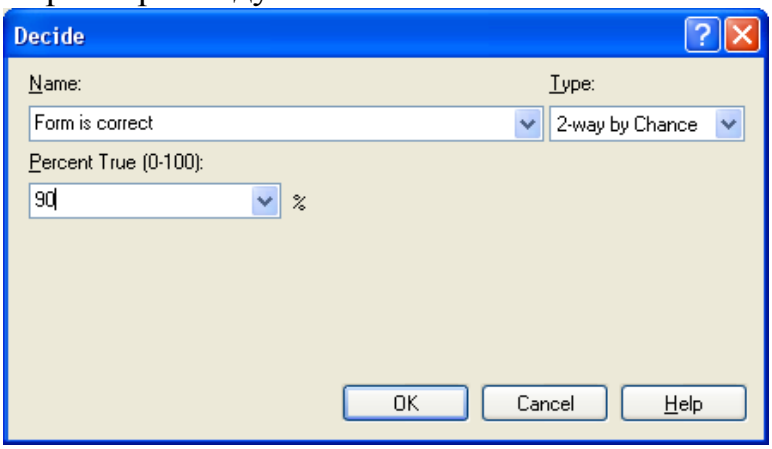
Обозначение:



Таким образом, модуль Decide позволяет реализовать модели с ветвлением, реализовать какой-то условие. Отметим, что прохождение транзактом модуля Decide происходит мгновенно, не занимает модельного времени [13].

Существует два варианта модуля Decide – «По вероятности» (by chance) или по условию (by condition). Тип задается в параметрах модуля.

Параметры модуля:

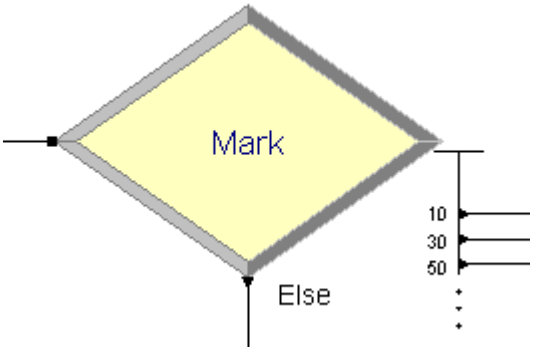


Выбран вариант «по вероятности». Задается вероятность того, что транзакт последует по правой ветке в процентах (Percent True). В данном случае, документ заполнен правильно с вероятностью 90%.

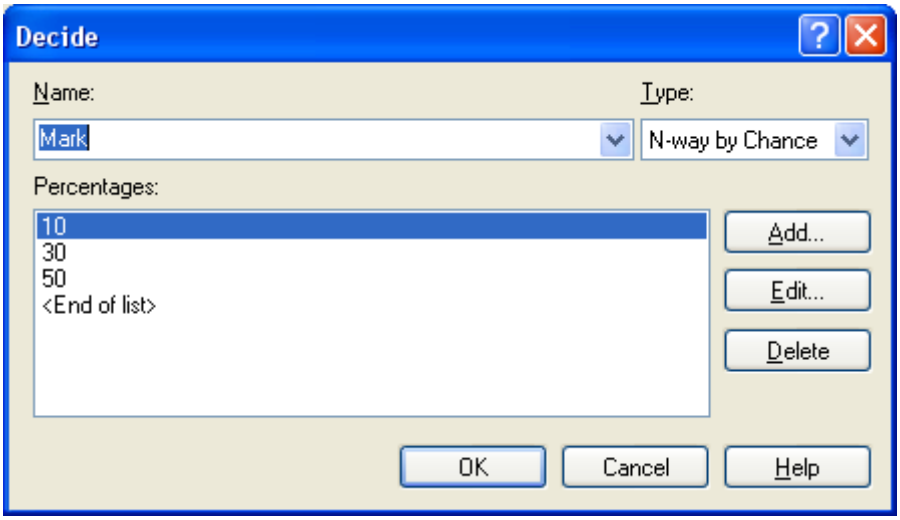
Другие примеры с модулем Decide «по вероятности»: деталь оказалась бракованной; клиент нуждается в дополнительных услугах; автомобилю нужно свернуть налево и т.д.

Модуль Decide может иметь не две, а несколько исходящих ветвей. Это относится как к варианту «по условию», так и к варианту «по вероятности». На рисунке показан случай, когда студент может получить одну из четырех оценок, от чего будет зависеть дальнейшая обработка соответствующего транзакта.

[13] При описании модулей мы будем явно говорить, будет ли в нем задержан транзакт. Если такого указания нет – значит модуль проходится мгновенно.



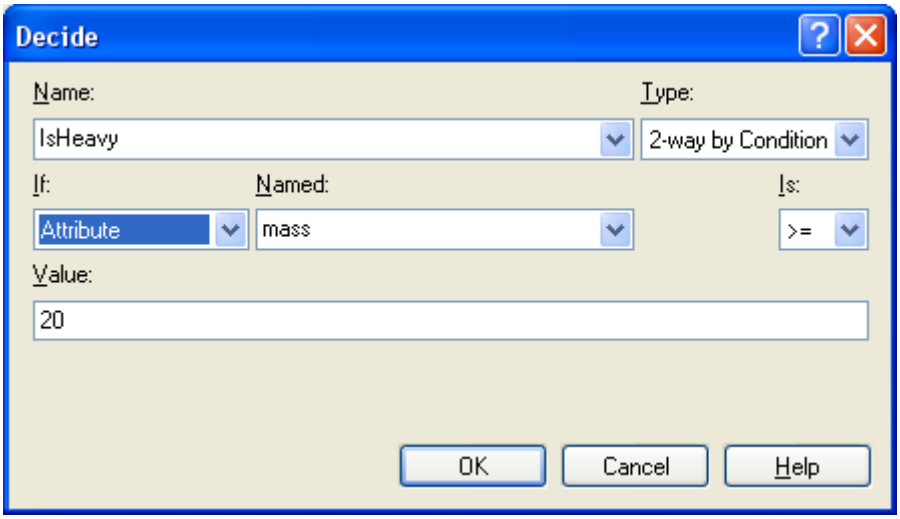
Окно параметров приведено на следующем рисунке. Выбран вариант «N-way by Chance», то есть несколько вариантов с вероятностью. Теперь необходимо задать вероятность в процентах для всех правых ветвей. Само собой, сумма должна быть меньше 100%. В нашем примере, вероятность получить «отлично» составляет 10%, «хорошо» – 30%, «удовлетворительно» – 50%, «неудовлетворительно» – оставшиеся 10% (нижняя ветвь Else).



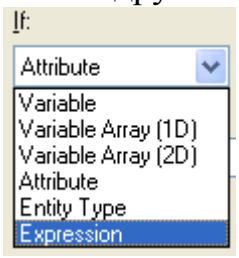
Тип модуля Decide «По условию» («by condition») позволяет направлять транзакты в зависимости от значения некоторого логического выражения истина/ложь. Например: машины с весом, превышающим некоторый лимит, направляются на объездную дорогу; если какое-то устройство занято, агрегат направляется на резервное устройство, документ определенного типа пересылается заданному исполнителю и т.д.

В примере на следующем рисунке предполагается, что у транзакта-грузовика [14] имеется атрибут *mass*. Если значение этого атрибута больше или равно 20, транзакт перейдет по ветке True (правой), иначе по ветви False (нижней).

[14] Речь идет транзакте, который в настоящий момент проходит модуль.



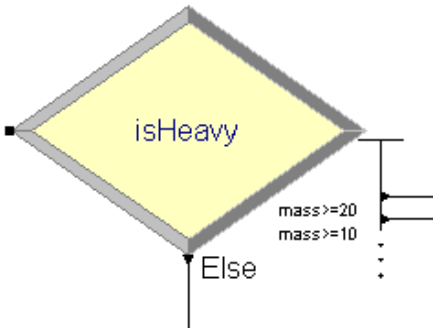
Есть и другие варианты условия:



В модуле decide может проверяться:

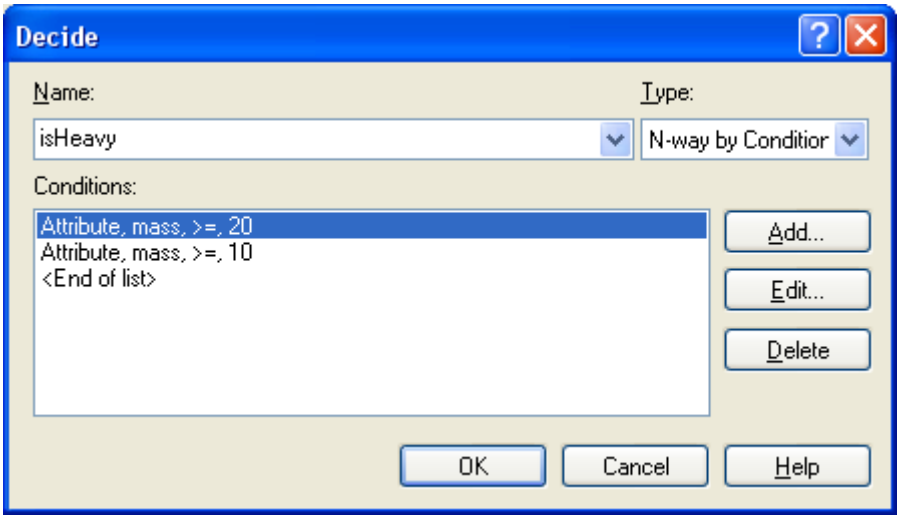
- значение переменной (Variable);
- значение элемента массива (Variable Array 1D, Variable Array 2D);
- значение атрибута транзакта (Attribute);
- тип транзакта (Entity Type);
- произвольное выражение (expression).

Для блока Decide с типом «по условию» также возможно наличие нескольких выходов.



Немного исправим пример с грузовиками, теперь, машины с массой 20 тонн и больше направляются по одному пути, от 10 до 20 тонн – по второму, меньше 10 тонн – по третьему.

Параметры модуля Decide показаны на рисунке:



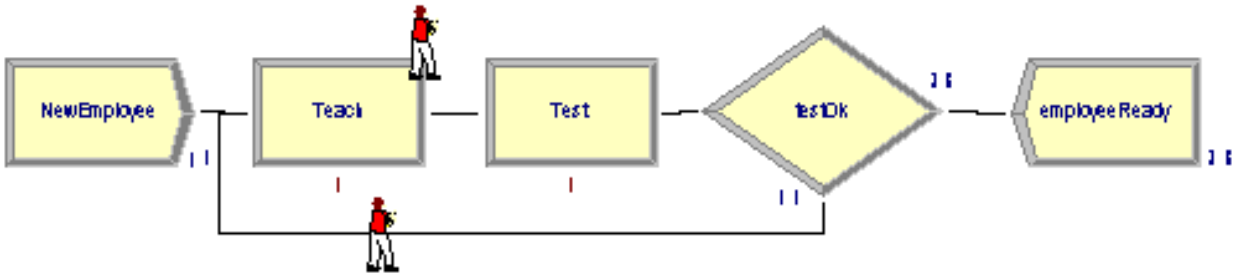
Необходимо указать условия (они могут быть разного типа) для каждой из правых веток. Для этого используются кнопки Add, Edit, Delete. При этом будет открываться окно задания условий.

Обратим внимание на порядок условий в примере. Если бы мы поменяли их местами, а пришедший транзакт представлял бы грузовик весом 25 тонн, то первое условие было бы выполнено (атрибут *mass >= 10*) и грузовик отправился бы на неверный путь. Таким образом, следует внимательно следить за порядком условий.

Рассмотрим теперь несколько примеров моделей с модулем Decide.

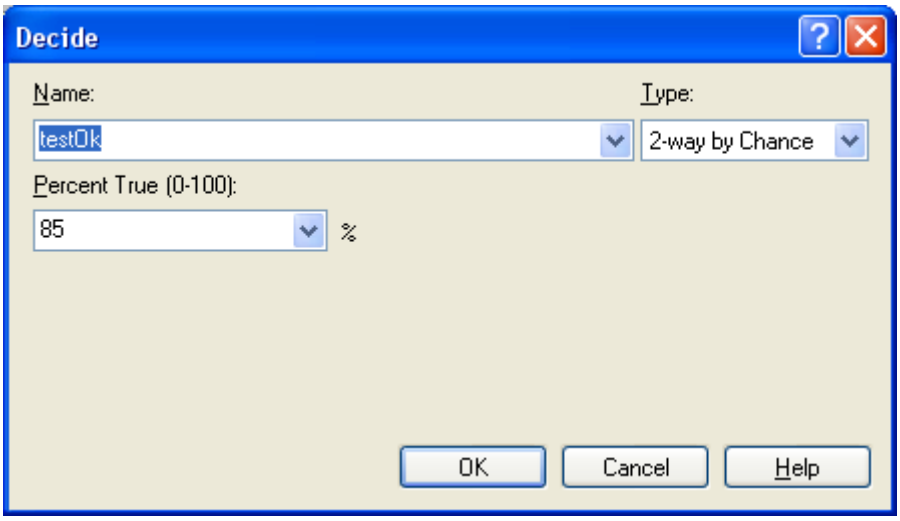
Пример 3. Новые работники поступают на инструктаж по технике безопасности. После инструктажа следует небольшой тест, и если работник с ним не справляется (вероятность этого 15%), работник должен повторить инструктаж и тест снова.

Граф модели показан на рисунке:



Понятно, что модули Process с названиями Teach и Test соответствуют операциям обучения и проверке знаний. Далее следует модуль Decide, распределяющий сотрудников в зависимости от результатов теста. Не прошедшие тест сотрудники (нижняя ветка false) вновь отправляются на модуль процесса Teach.

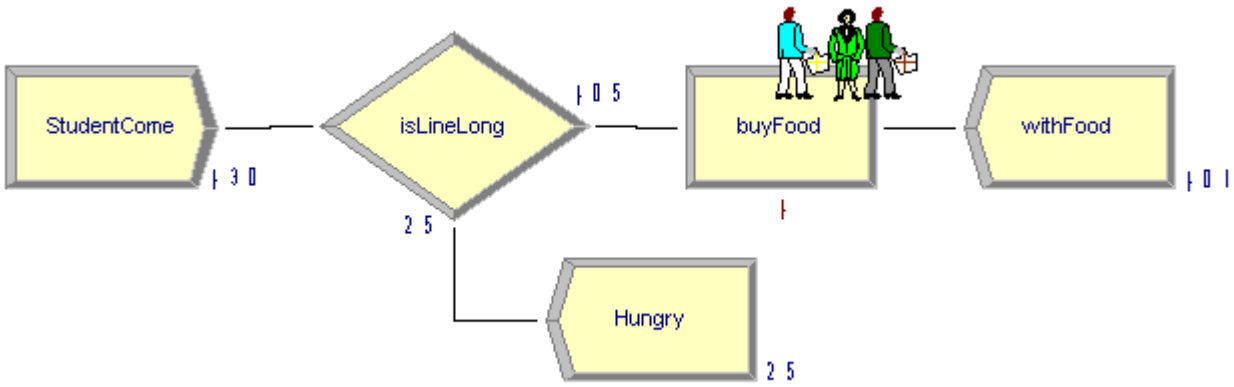
Параметры модуля Decide:



Как видим, выбран тип «по вероятности» и указана вероятность успешного прохождения теста.

Пример 4. Студенты посещают буфет. Если студент обнаруживает, что очередь превышает 5 человек, то он уходит.

Граф модели показан на рисунке:



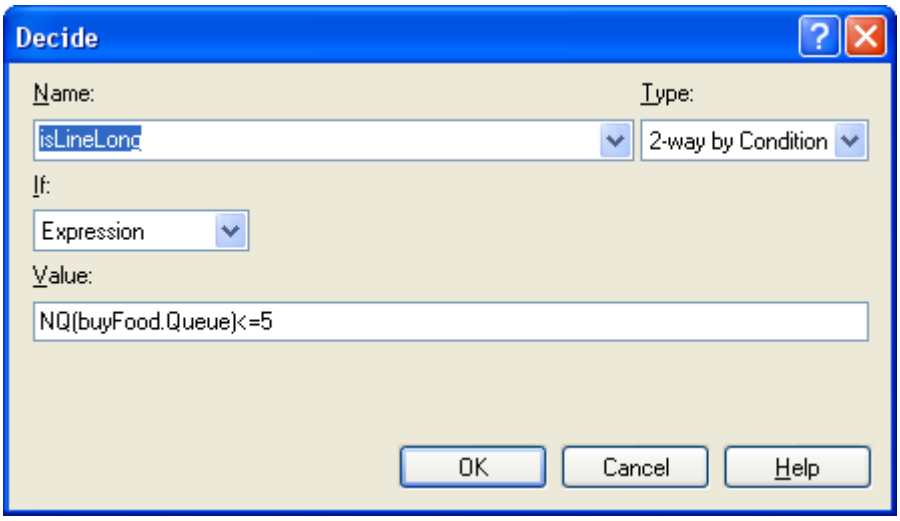
Структура модели очевидна. Модуль процесса buyFood соответствует покупке продуктов в буфете, и требует ресурса – продавца. Таким образом, у модуля будет очередь для студентов, ожидающих обслуживания.

Два модуля Dispose позволяют оценить, сколько студентов смогли купить себе обед (withFood), а сколько – нет (Hungry).

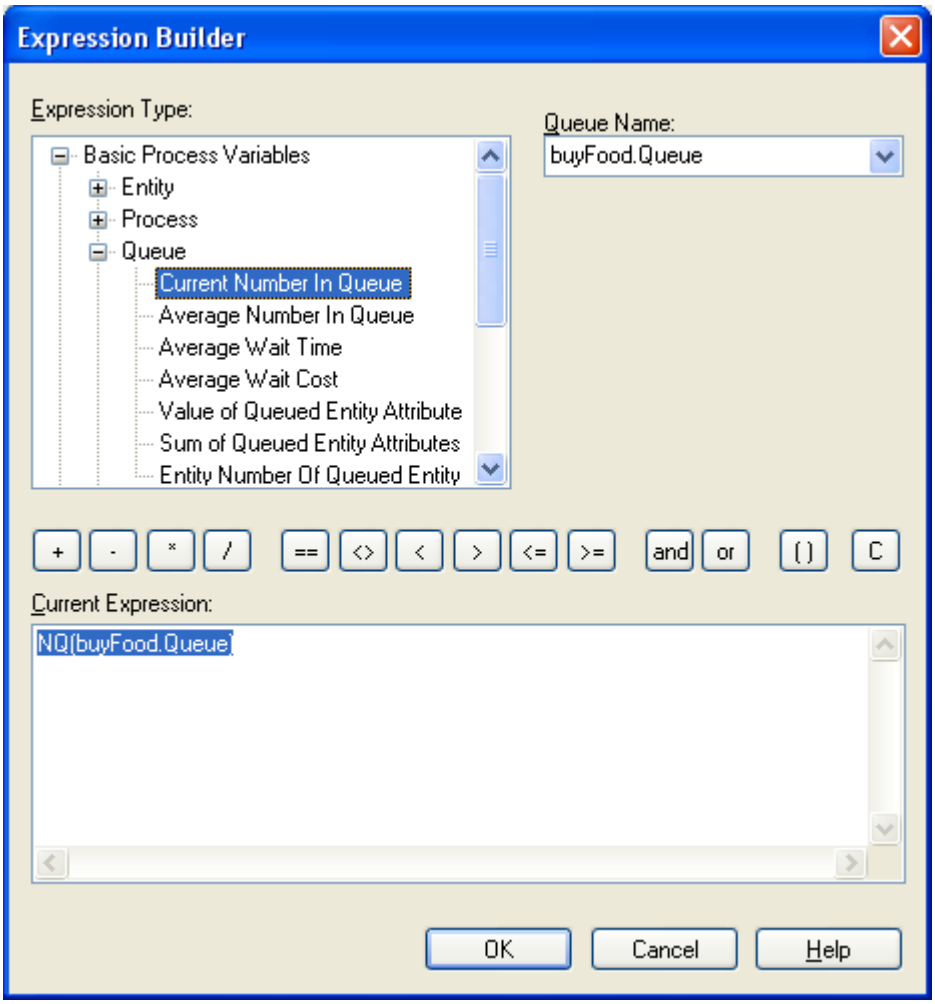
Рассмотрим модуль Decide с названием IsLineTooLong. Здесь выбран тип «По условию». Условием является выражение «Expression». Задано следующее логическое выражение:

NQ(buyFood.Queue)<=5

То есть проверяется, что число транзактов в очереди с именем buyFood.Queue в настоящий момент (функция *NQ()*) меньше или равно пяти.



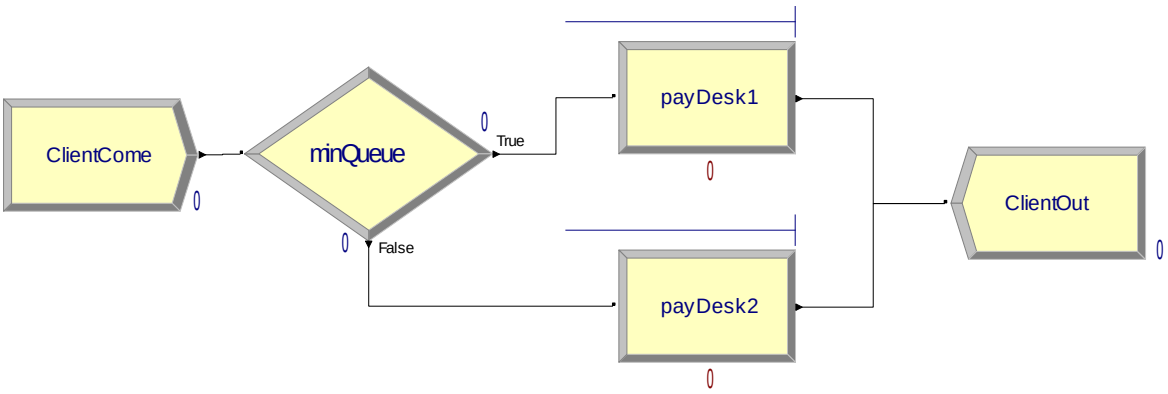
Для ввода этого выражение удобно использовать построитель выражений. Если в поле Value нажать правую кнопку мыши и в контекстном меню вызвать Build expression, то в удобном окне построителя выражений можно легко получить доступ к необходимым объектам Arena:



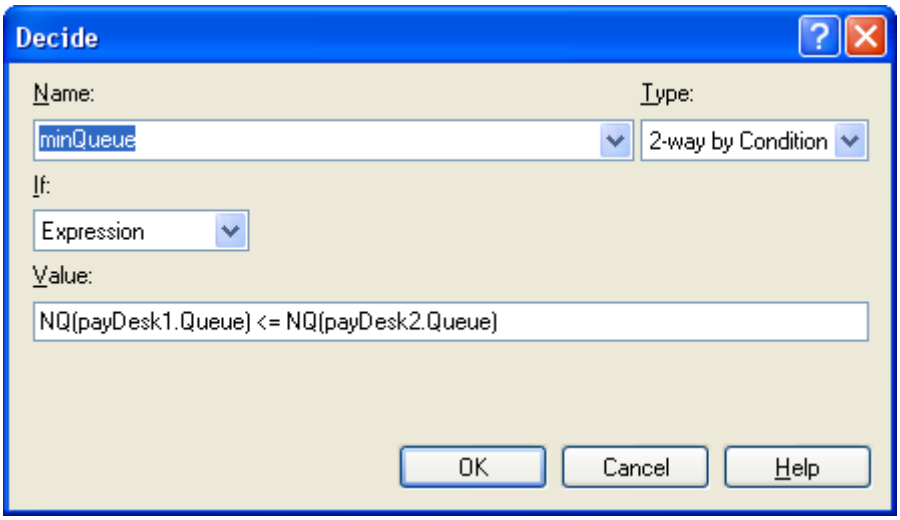
В данном случае в древоподобном списке мы выбрали Basic Process Variables (что дало нам доступ к объектам Arena), затем Queue (очереди), а затем функцию Current Number in Queue (обозначение *NQ()*). В выпадающем списке справа мы выбрали имя интересующей нас очереди buyFood.queue.

Рассмотрим еще одну распространенную ситуацию, когда выбор дальнейшего пути зависит от состояния различных компонентов системы. В таких случаях применяется модуль Decide по условию и различные функции, характеризующие состояние объектов.

Покупатель в супермаркете станет выбирать кассу с более короткой очередью.



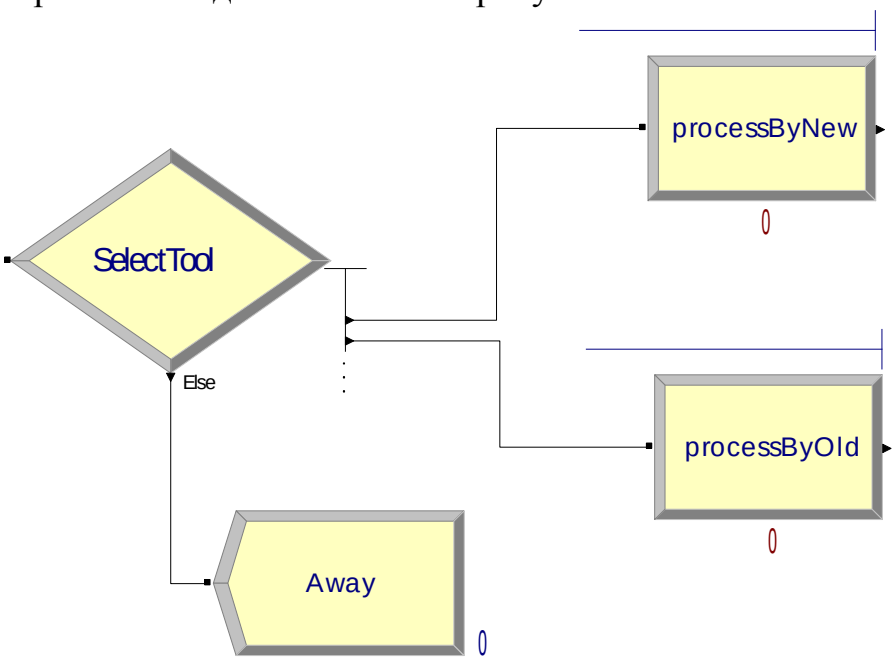
В модуле decide происходит выбор дальнейшего пути покупателя. Если очередь у первой кассы короче – клиент направляется к ней, иначе – ко второй кассе.



Рассмотрим еще один пример.

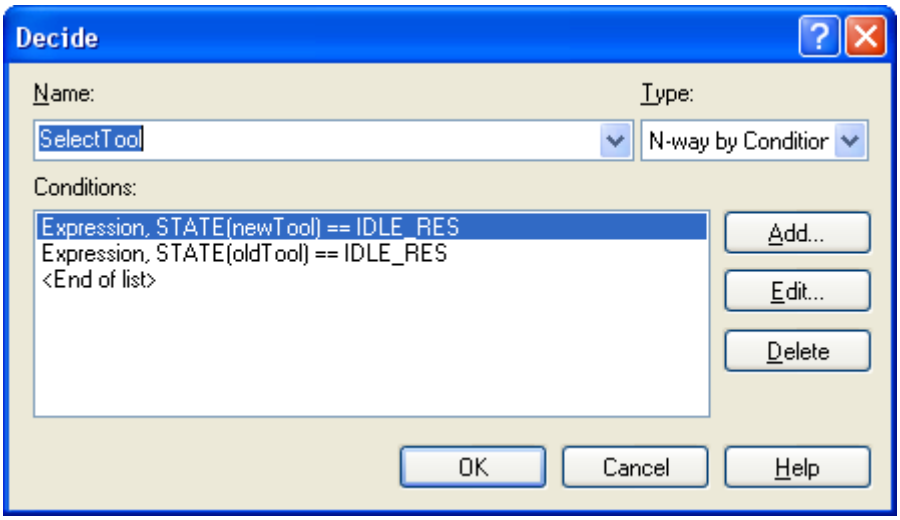
Пример 5. Поступившая деталь будет проходить обработку на более современном станке, но если этот станок занят – деталь будет обработана на менее современном. Если заняты оба станка, деталь будет выведена из цеха без обработки.

Фрагмент модели показан на рисунке:



Имеются два ресурса newTool и oldTool, они используются в соответствующих процессах. Оба модуля имеют очереди, однако в ходе моделирования эти очереди всегда будут пусты.

Параметры модуля Decide:



Модуль имеет тип «N-way by condition». Имеются три выхода – на новый станок, на старый станок и на выход без обработки (else, по умолчанию). Для первых двух исходящих веток прописаны условия. Для нового станка:

STATE(newTool) == IDLE_RES

Для старого станка:

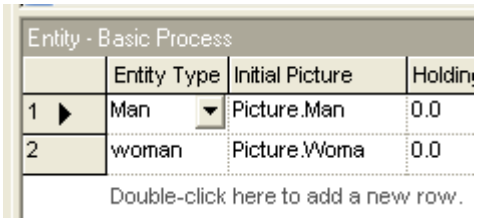
STATE(oldTool) == IDLE_RES

Функция *STATE()* возвращает состояние заданного ресурса. Константа *IDLE_RES* означает, что ресурс свободен.

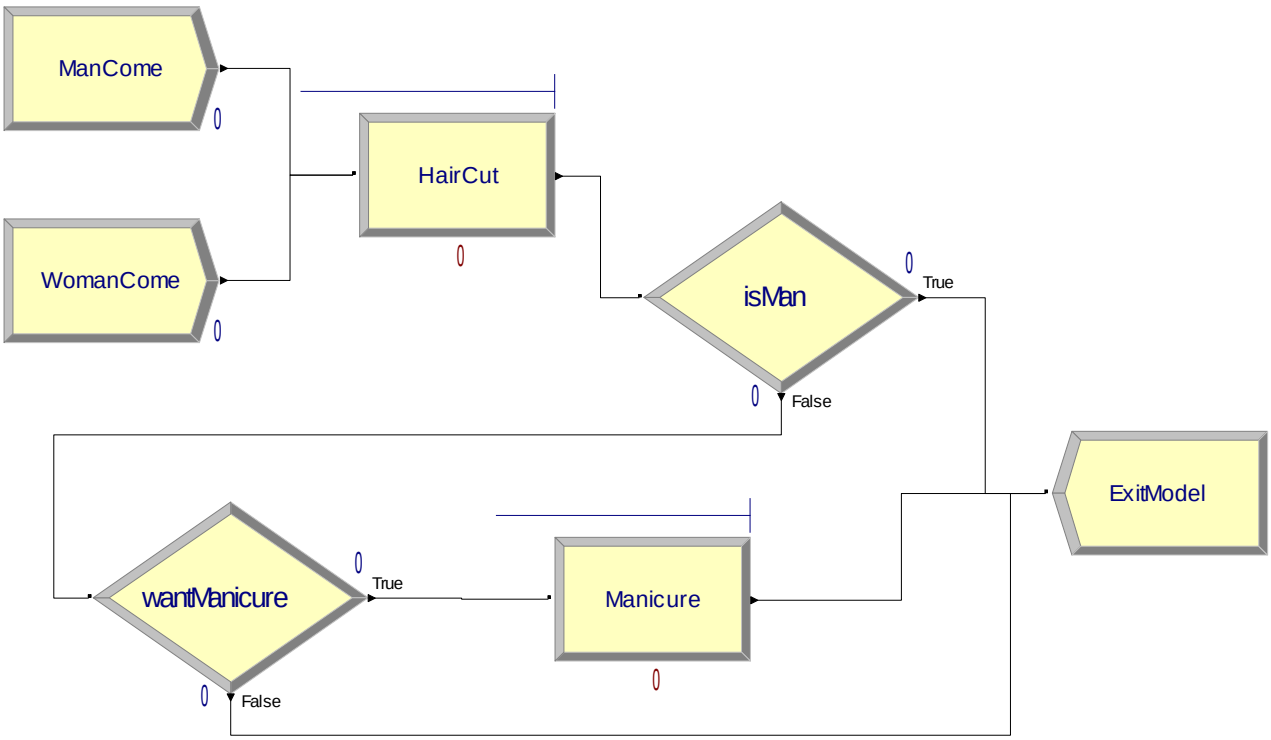
Обратим внимание на порядок условий. Сначала проверяется состояние нового станка, поскольку именно он является по условию задачи предпочтительным.

Пример 6. Вернемся теперь к примеру с парикмахерской (**пример 1**). Внесем в него исправления, будем считать, что прибывают клиенты двух типов – мужчины и женщины. Причем женщина с вероятностью 25% помимо стрижки захочет сделать маникюр, за что отвечает специальный сотрудник. Продолжительность маникюра – 30± 5 минут [15].

Теперь у нас два типа транзактов, с соответствующими изображениями:

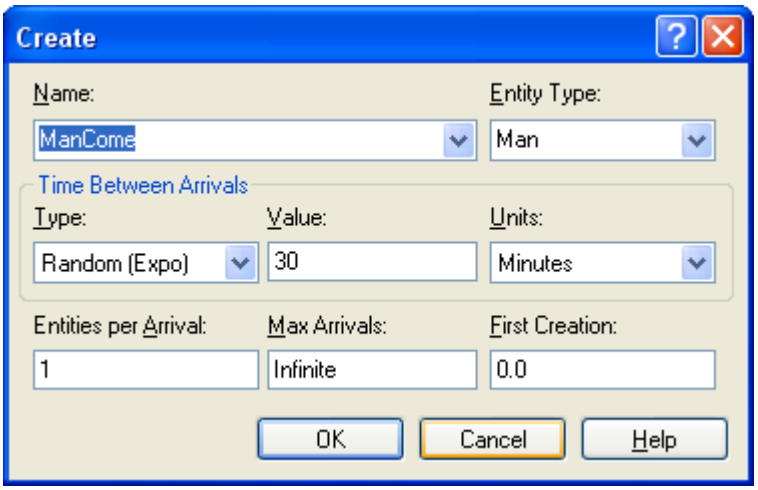


На рисунке представлен новый граф модели:



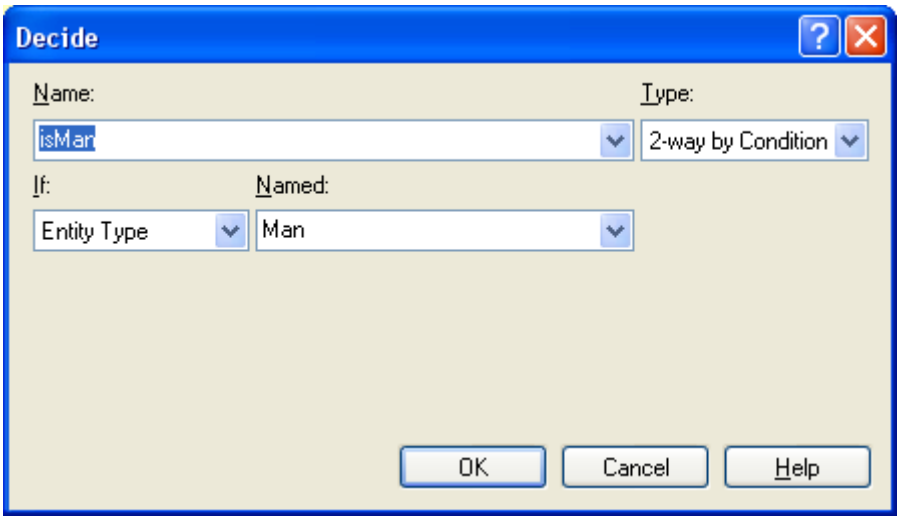
[15] В полученную модель заложены следующие предположения: длительность стрижки мужчин и женщин одинакова; одни и те же мастера могут стричь как мужчин, так и женщин. Подобные предположения остаются в моделях всегда и ограничивают их использование. Исследователю крайне важно учитывать все закладываемые допущения и оценивать их критичность и корректность.

Генераторы транзактов ManCome и WomanCome вводят в модель соответственно транзакты «мужчина» и «женщина»:

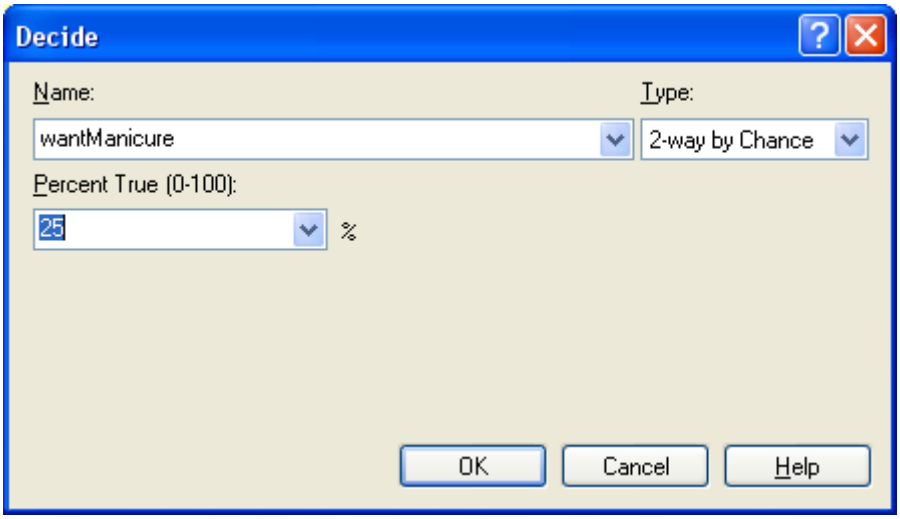


Далее эти транзакты направляются в модуль Process, отвечающий за стрижку. После модуля стрижки имеется модуль isMan типа Decide, который направляет транзактов-мужчин и транзактов-женщин в разные ветви. Транзакт-мужчина покидает модель с помощью модуля ExitModel типа Dispose, а транзакт-женщина направляется далее в модуль WantManicure.

В параметрах модуля isMan используется тип «by condition» (по условию), причем в качестве условия выбран тип транзакта Entity type. Если транзакт – мужчина, он направляется по правой ветке (True):



Транзакты-женщины попадают во второй модуль Decide с названием WantManicure, где определяется (по вероятности), желает ли женщина сделать маникюр.



С вероятностью 25% транзакт перейдет по правой ветке в модуль Process соответствующий маникюру Manicure. В противном случае, транзакт покидает модель.

Задача 1. Посетители небольшого магазина самообслуживания приходят раз в 3 минуты, закон распределения – экспоненциальный. Покупатели тратят на выбор товара от 3 до 7 минут, закон распределения равномерный. Обратим внимание, что одновременно выбирать товар возле прилавков может неограниченное число покупателей. Затем покупатели отправляются к кассам. В магазине две кассы и покупатель перейдет к той, возле которой меньше очередь. Обслуживание на кассе занимает от 3 до 5 минут, закон распределения равномерный. Построить модель магазина, оценить среднее время ожидания в очереди, загрузку кассиров.

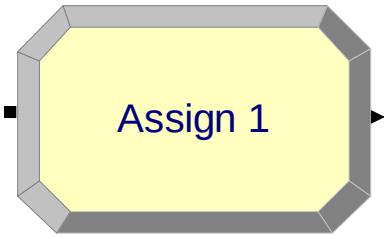
Контрольные вопросы

1) Для чего предназначен модуль Decide?
2) В чем разница между вариантами модуля Decide «по условию» и «по вероятности»?
3) Каким образом можно организовать направление транзакта в модели в зависимости от характеристик транзакта?
4) Как проверить длину определенной очереди?
5) Каким образом можно направить транзакт в ту или иную ветвь при условии занятости определенного ресурса?
6) Как можно направить транзакты разных типов в разные модули?
7) Почему важна правильная последовательность условий для модуля с несколькими исходящими ветвями?
8) Можно ли использовать атрибуты транзакта в условии модуля Decide?
9) В каком случае модуль Decide может иметь несколько выходов?
10) Имеется ли возле модуля Decide очередь?
11) Что означают цифры, которые появляются у выходов модуля Decide при моделировании?
12) В каком случае в модуле Decide необходимо использовать логическое выражение?

4 Атрибуты, очереди, приоритеты

Модуль **Assign** обеспечивает присвоение значений переменным модели и атрибутам транзакта, проходящего через модуль. Может выполнять сразу несколько присвоений.

Обозначение:



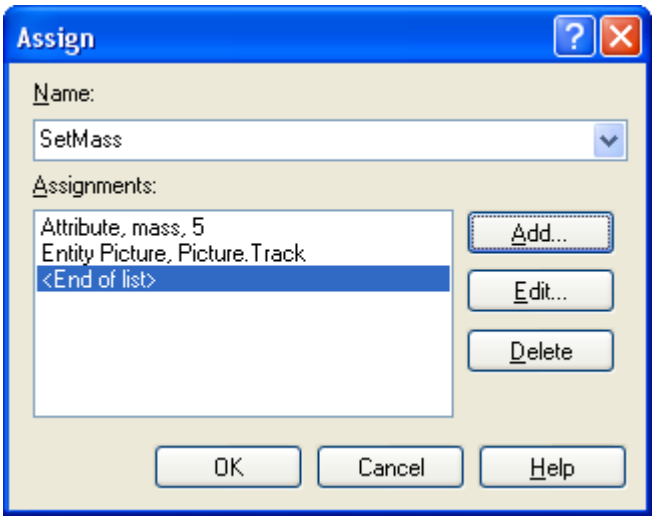
Модуль можно использовать для разных целей:

- задать какое-то свойство транзакта, от которого будет зависеть дальнейшая обработка этого транзакта;
- «пометить» транзакт, например, показав, что он прошел какую-то стадию обработки;
- поменять тип транзакта или его картинку;
- работать с переменными, например, собирая различную статистику;

и т.д.

Например, имеется переменная «Баланс» и транзакты-документы. Приходные документы увеличивают баланс, расходные уменьшают. Переменные также могут задавать какие-то условия работы модели, например, если на предприятие происходит плановая профилактика, некоторые процессы можно приостановить. Еще один популярный способ использования модуля – задание атрибутов приоритета, которые работают вместе с очередью.

Пример параметров модуля Assign показан на рисунке.

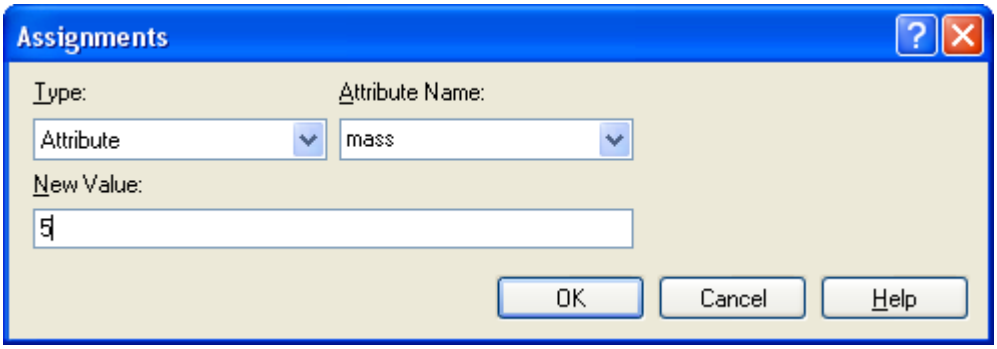


Модуль может содержать несколько назначений. В данном случае, мы устанавливаем для проходящего транзакта атрибут *mass*, присваивая ему

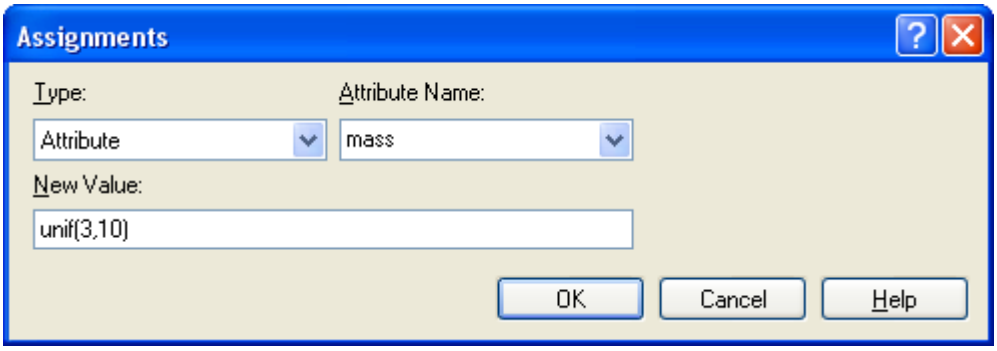
значение 5 тонн. А также меняем для транзакта картинку, изменив специальный атрибут *Entity Picture*.

Кнопками Add, Edit и Delete можно вести список присвоений. При этом открывается специальное окно, где указывается тип присвоения, имя переменной или атрибута, присваиваемое значение.

На рисунке показано присвоение значения атрибуту транзакта с именем *mass*:

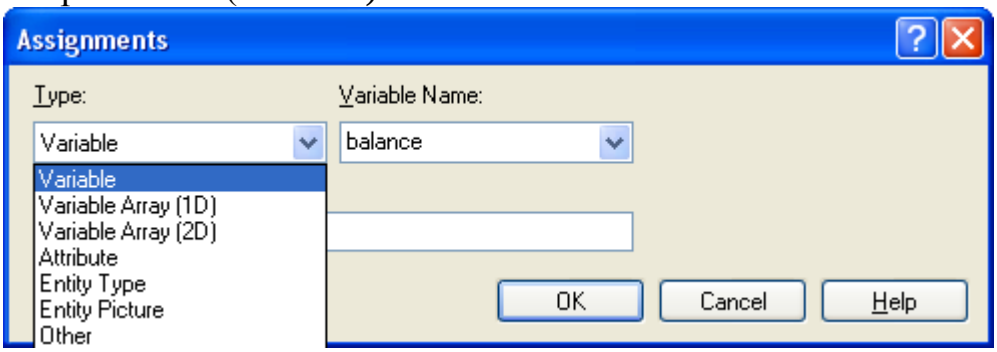


Можно использовать выражение, например, указать, новое значение случайным образом. На следующем рисунке атрибут получит случайное значение, от 1 до 10, закон распределения - равномерный.

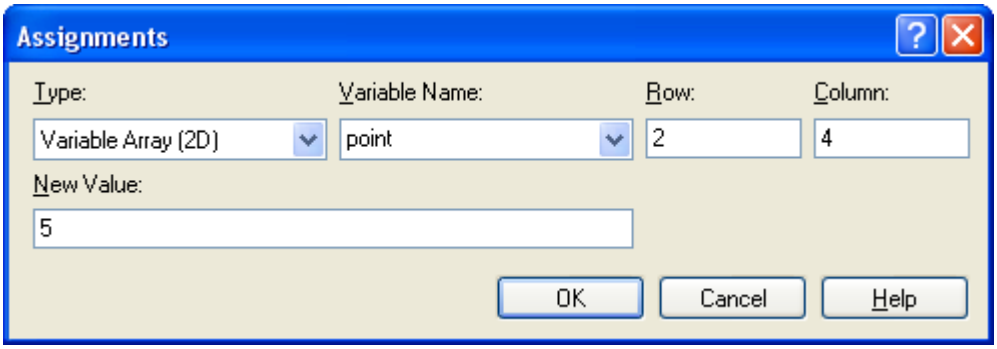


В модуле Assign значение может присваиваться:

- переменной (Variable):



- элементу двумерного или одномерного массива (Variable Array):



- атрибуту транзакта (Attribute);
- типу транзакта (Entity Type);
- картинке транзакта (Entity Picture);
- произвольному объекту системы (Other).

Рассмотрим пример модели.

Пример 7. Построить модель обработки документов (счёт - фактур и выписок банка) бухгалтерией предприятия (3 бухгалтера) в программе «1С:Бухгалтерия». Параметры процесса, представлены в таблице. Обработка выписки банка имеет приоритет перед обработкой счета-фактуры.

В результатах моделирования будет получена следующая информация:

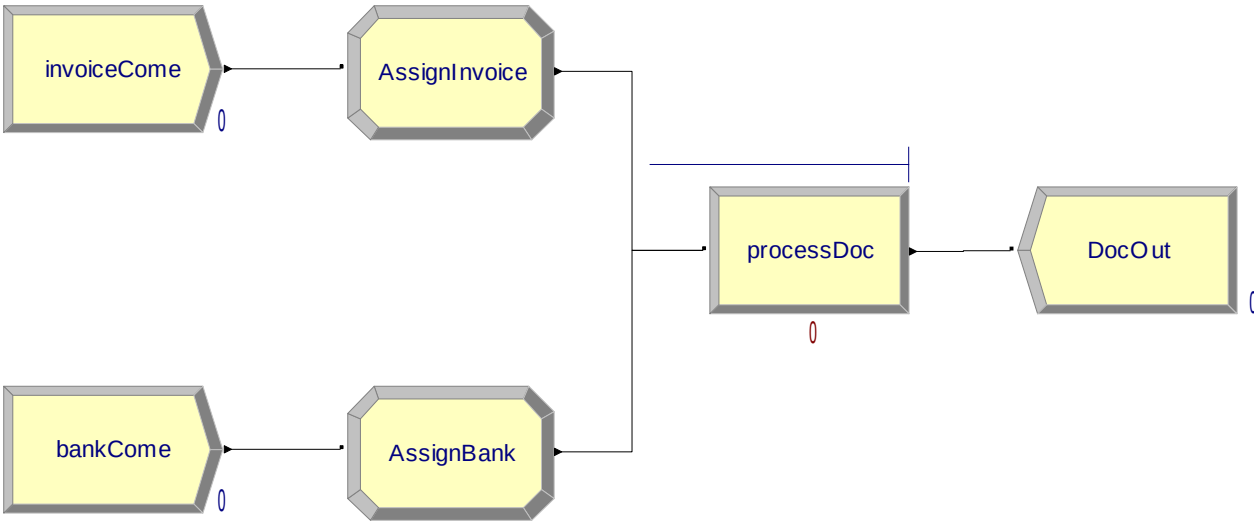
- число документов (счёт - фактур и выписок банка по отдельности), поступивших в бухгалтерию в течение контрольного периода времени;
- число обработанных документов (счёт - фактур и выписок банка по отдельности);
- число необработанных документов (счёт - фактур и выписок банка отдельности);
- среднее время ожидания обработки счёт - фактур и выписок банка;
- процент загрузки бухгалтерии.

Величина	**Закон распределения**	**Параметры**
Интервал прихода выписки банка	равномерный	мат. ожидание: 20 минут
		ср. кв. отклонение: 4 минут
Интервал прихода счёт - фактуры	нормальный	мат. ожидание: 10 минут
		ср. кв. отклонение: 2 минут
Время обработки бухгалтером счёт - фактуры	экспоненциальный	мат. ожидание: 20 минут
Время обработки бухгалтером выписки банка	экспоненциальный	мат. ожидание: 15 минут

Решим задачу с помощью одного модуля Process и двух модулей Assign[16]. Имеется два транзакта: счет-фактура и выписка банка и один ресурс – бухгалтер с мощностью 3.

[16] Предлагаемый вариант решения не является единственно возможным.

Граф модели показан на рисунке:

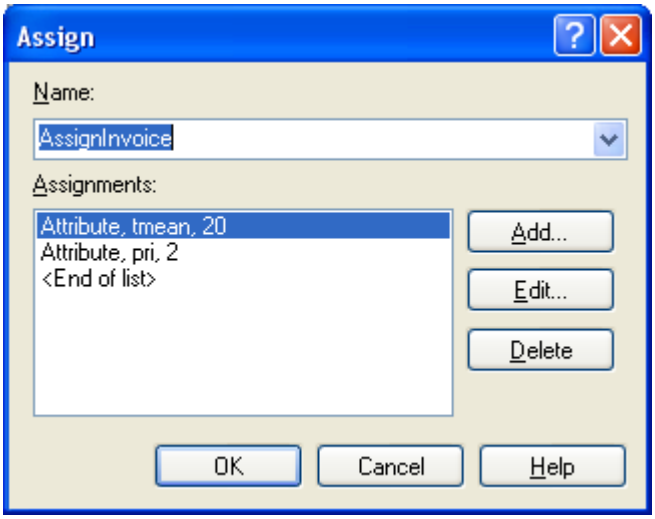


Таким образом, в модели два генератора транзактов для каждого из типа документов, а также два модуля Assign, которые призваны указать атрибуты для приходящих документов.

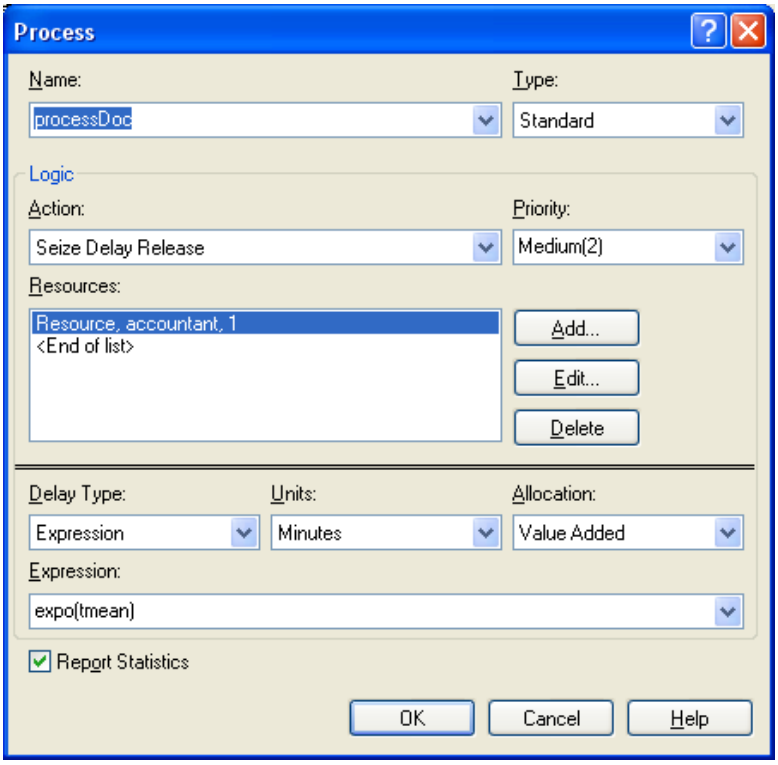
Будем использовать два атрибута:

Название	Назначение	Значение для выписки	Значение для счет-фактуры
pri	Значение приоритета	1	2
tmean	Среднее время обработки документа	15	20

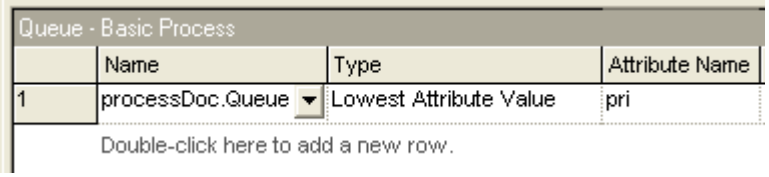
Присвоение атрибутов с помощью модуля Assign показано на рисунке:



Модуль Process использует значение атрибута *tmean* при расчете времени обработки транзакта – задано выражение *expo(tmean)*. Теперь счет-фактура будет обрабатывать в среднем за 20 минут, а выписка банка – за 15.

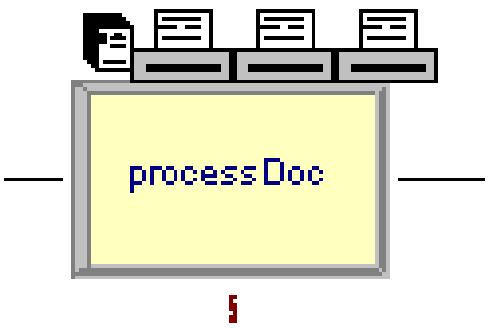


Теперь обратимся к свойствам очереди ProcessDoc.Queue. Это можно сделать с помощью невизуального модуля Queue:



Мы видим, что установлен тип очереди *«По наименьшему значению атрибута»* и указано имя атрибута *pri*. Поскольку для выписок банка значение атрибута *pri* меньше, в очереди эти документы будут пропускаться вперед. Если в очереди есть только счета-фактуры, новая выписка будет помещена в начало очереди. Если имеются документы обоих типов – новая выписка попадет в позицию после последней имеющейся выписки и перед первой счет-фактурой.

На рисунке выписки банка показаны в виде картинки факса, а счет-фактуры – в виде бумажного документа.



Работа с очередями с использованием модулей Assign может использоваться и для задач, в которых приоритет отдается по какому-то содержательному признаку транзакта. Например, нужно пропустить больного с более тяжелым состоянием; оплатить сначала счета с меньшей суммой; отгрузить в первую очередь более массивные контейнеры и т.д.

К сожалению, существующие модули системы Arena не предусматривают реализацию «абсолютного приоритета», когда вновь прибывший транзакт может прервать обслуживание транзакта с более низким значением приоритета. Например, пришедший телефонный звонок может заставить работника приостановить работу над документом; поступление экстренного больного может приостановить осмотр врачом планового пациента; случившаяся неполадка компьютера приостанавливает выполнение работ и т.д.

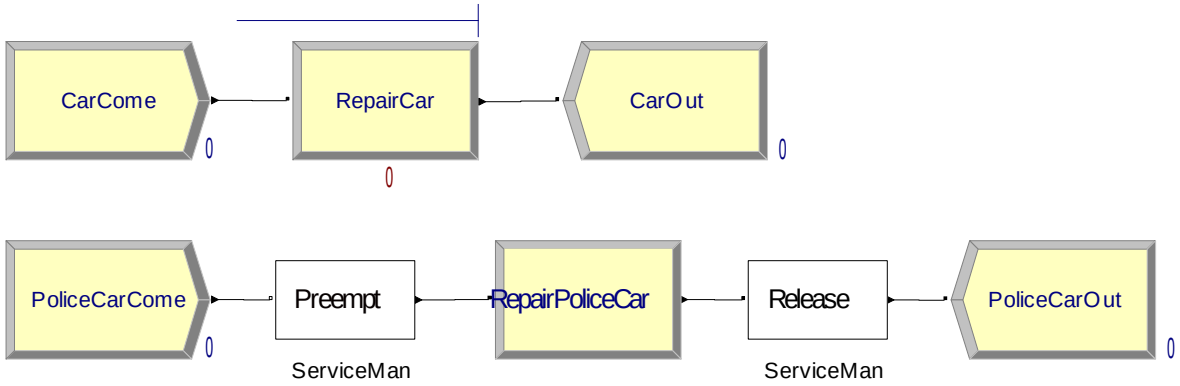
Для решения подобных задач требуется обращаться непосредственно к языку моделирования SIMAN и к его блокам. Визуальные представления блоков содержаться в шаблоне Blocks [17]. В отличие от модуля, блок точно соответствует одной команде языка SIMAN.

В данном случае нам потребуются два блока: PREEMPT, который приостанавливает обслуживание транзакта с помощью некоторого ресурса и переключает этот ресурс на проходящий транзакт. А также блок RELEASE, освобождающий ресурс (в принципе можно было бы использовать и модуль Release).

Пример 8. На станции технического обслуживания автомобилей работает один специалист по ремонту. Раз в два часа (закон распределения экспоненциальный) прибывает очередная машина. Раз в 4 часа (закон распределения экспоненциальный) прибывает полицейский автомобиль. В этом случае работник приостанавливает обслуживание обычного автомобиля и приступает к работе с полицейской машиной. Затем он возвращается к прерванной работе и дообслуживает транзакт (обычную машину). Ремонт любой машины, как обычной, так и полицейской требует 1-2 часа (закон распределения равномерный).

Граф модели изображен на рисунке.

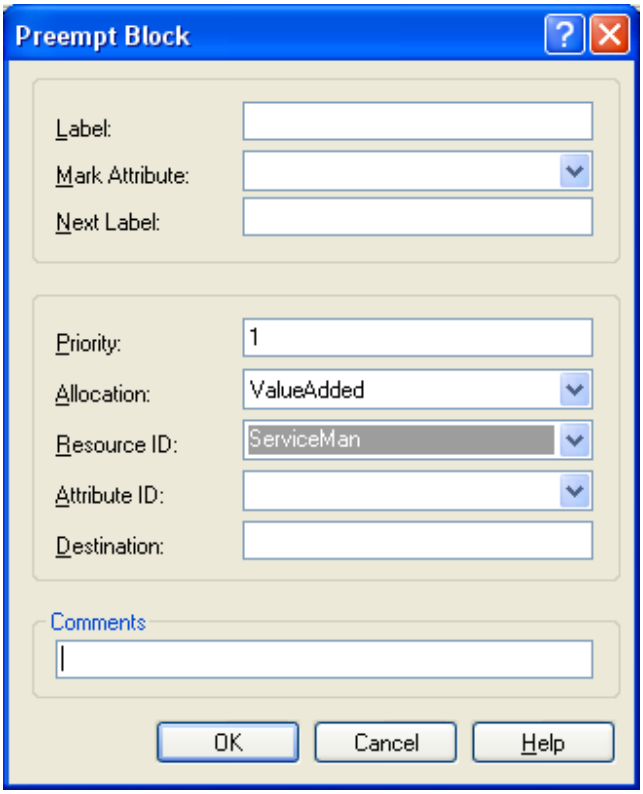
Верхний сегмент не требует пояснений, он отвечает за обработку обычных машин.



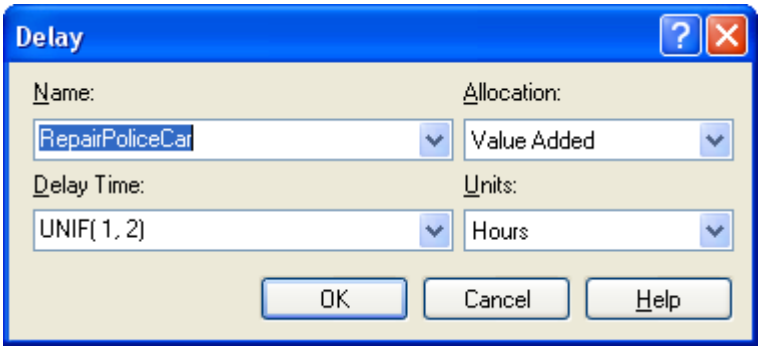
[17] Каждый шаблон объединяют ряд визуальных и невизуальных модулей определенной направленности. Чтобы подключить шаблон необходимо в левой стороне окна нажать правую кнопку мыши, выбрать в появившемся меню пункт Attach, и затем выбрать в диалоге открытия файла нужный файл шаблона.

Нижний сегмент отвечает за обработку поступающих полицейских машин.

Как только поступает транзакт, соответствующий полицейской машине, он попадает в блок PREEMPT, который обеспечивает перехват ресурса ServiceMan, который в это время мог быть занят обслуживанием обычной машины.



Далее следует обычный модуль Delay из шаблона Advanced Process, описывающий ремонт полицейской машины.

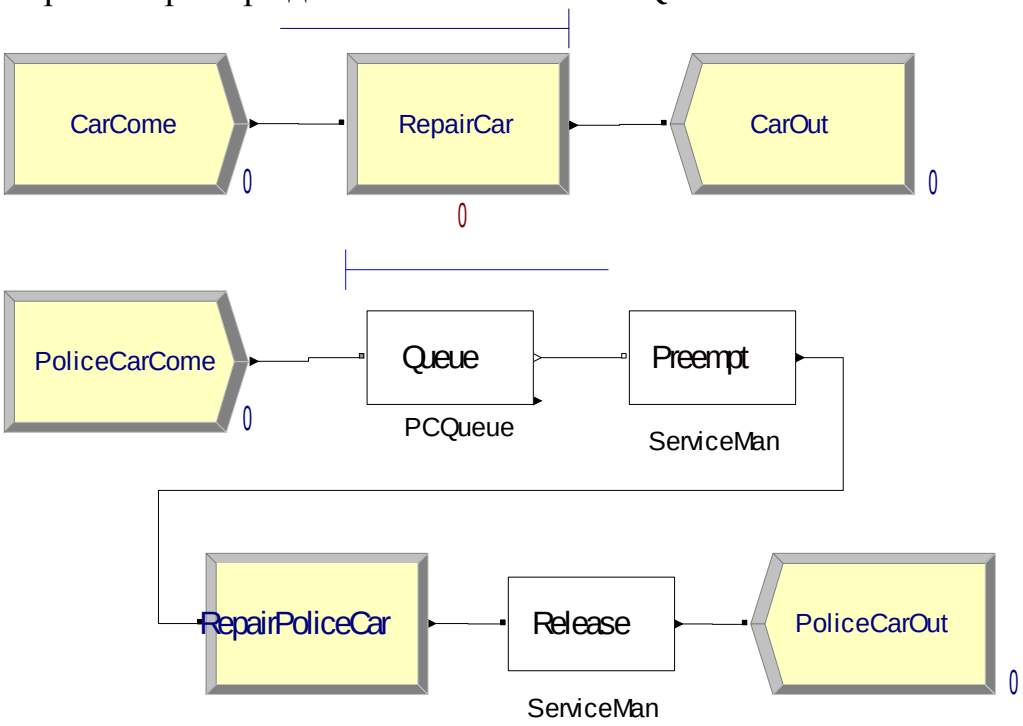


После этого, блок RELEASE отпускает ресурс ServiceMan. Теперь, если ранее работник прервал ремонт обычной машины, он сможет вернуться к этой работе и закончить ее.

Блок PREEMPT автоматически формирует очередь (если вслед за первой приходит вторая полицейская машина, ей приходится ожидать, пока будет закончен ремонт первой полицейской машины). Чтобы отразить эту очередь визуально, чтобы собирать по ней статистику при моделировании на Arena на уровне блоков необходимо явно помещать транзакт в очередь бло-

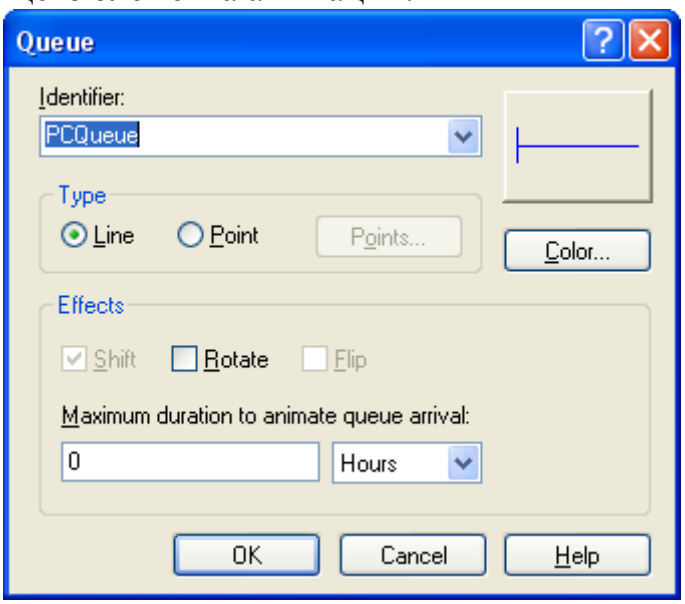
ком QUEUE. Если транзакт сможет пройти в следующий после блока QUEUE блок, то он не попадет в очередь. Иначе – будет задержан в этой очереди до тех пор, пока не сможет последовать в следующий блок (когда обслуживание предыдущей полицейской машины будет окончено).

Вариант примера дополненный блоком QUEUE:

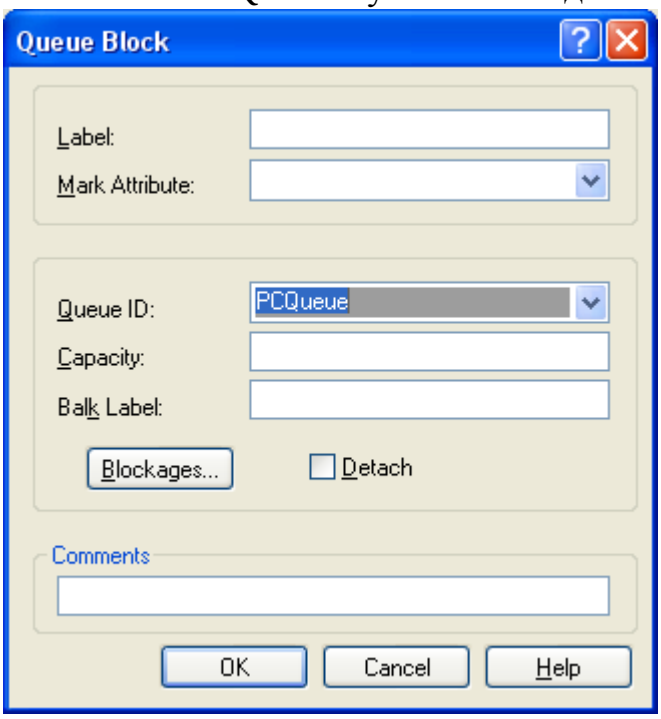


Саму очередь следует создать вручную в невизуальном компоненте QUEUE. Очередь для полицейских машин получила название PCQUEUE.

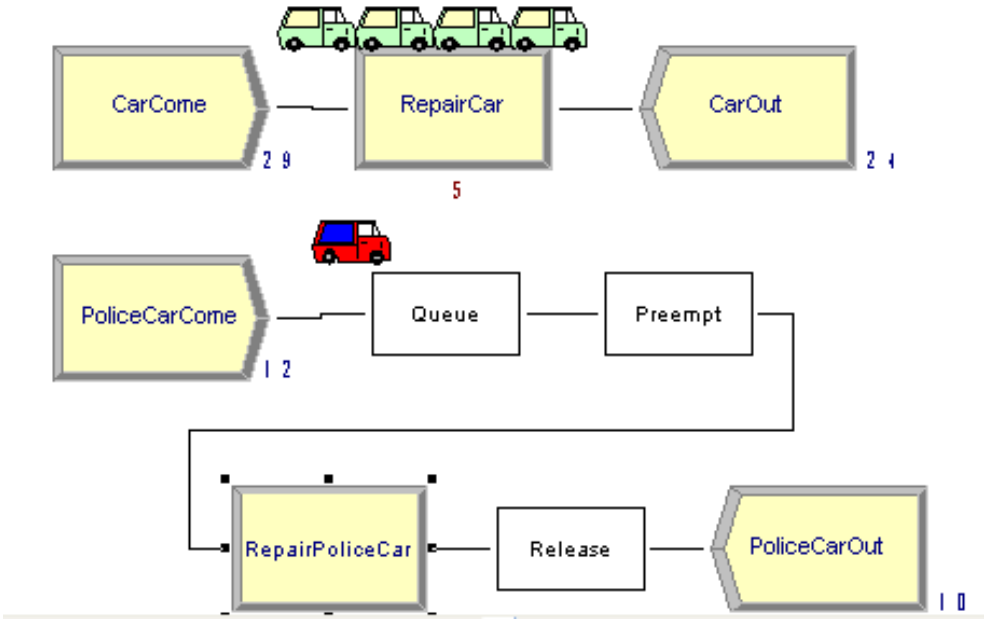
Далее потребовалось вручную разместить очередь с помощью соответствующего элемента анимации:



Затем в блоке QUEUE указываем идентификатор очереди PCQUEUE:



Теперь ожидающие ремонта полицейские машины помещаются в созданную очередь PCQUEUE, что позволяет отслеживать их визуально и собирать статистику.



Queue

Time

Waiting Time

	Average
PCQueue	1.2790
RepairCar.Queue	9.2261

Other

Number Waiting

	Average
PCQueue	0.1663
RepairCar.Queue	4.0965

Пример 9. Вернемся теперь к примеру с парикмахерской (**Пример 1, Пример 6**). Будем рассчитывать выручку заведения с учетом следующих расценок:

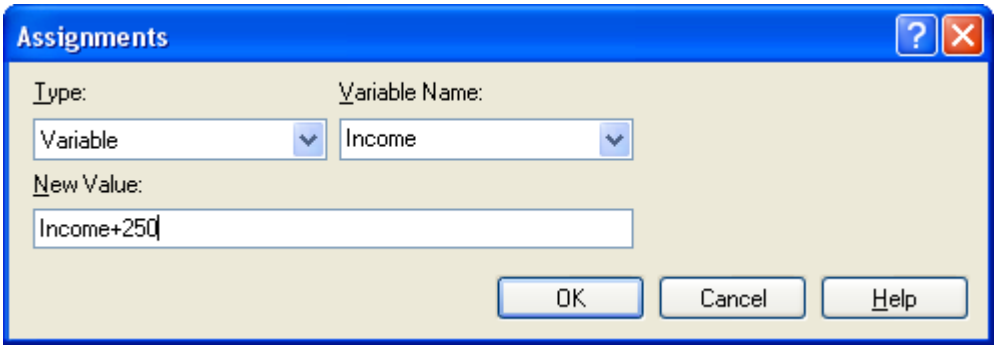
- мужская стрижка – 250 рублей;
- женская стрижка - 400 рублей;
- маникюр – 500 рублей.

Оценим выручку предприятия за период времени.

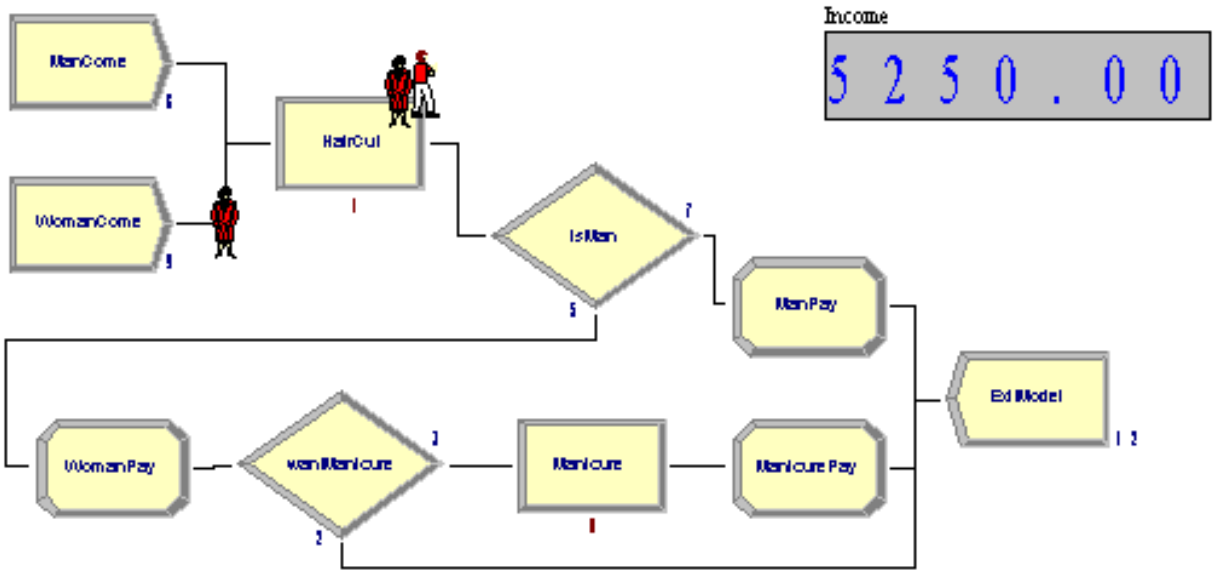
Для накопления информации о выручке заведем переменную *Income*. Для этого используем невизуальный модуль Variables, где введем имя переменной и зададим начальное значение – 0.

	Name	Rows	Columns	Data Type	Clear Option	File Name	Initial Values	Report Statistics
1	Income			Real	System		0 rows	
	Double-click here to add a new row.							

В граф модели поместим три модуля Assign, позволяющие учесть выручку от мужской стрижки, женской стрижки и маникюра. Например, в модуле с именем ManPay значение переменной *Income* увеличивается на 250.



Разместим графический элемент «переменная» и укажем в его поле expression имя переменной *Income*. Теперь по мере моделирования отображается получаемая выручка.



Пример 10. Усложним пример с парикмахерской (**Пример 1, Пример 6, Пример 9**). Будем считать, что женские прически в отличие от мужских могут быть нескольких видов, они будут отличаться по цене и по времени.

Типы причесок:

- «обычная», занимает 30+-10 минут, стоит 400 рублей;
- «сложная», занимает 60+-10 минут, стоит 700 рублей;
- «эксклюзивная», занимает 90+-10 минут, стоит 1200 рублей.

Будем считать, что женщины заказывают обычную прическу с вероятностью 50 %, сложную – 30%, эксклюзивную – 20%.

Заведем две переменных, точнее одномерных массива: *price* для цены и *tmean* для среднего времени стрижки.

Variable - Basic Process

	Name	Rows	Columns	Data Type	Clear Option	File Name	Initial Values
1	Income			Real	System		0 rows
2 ▶	Price	3		Real	System		3 rows
3	Tmean	3		Real	System		3 rows

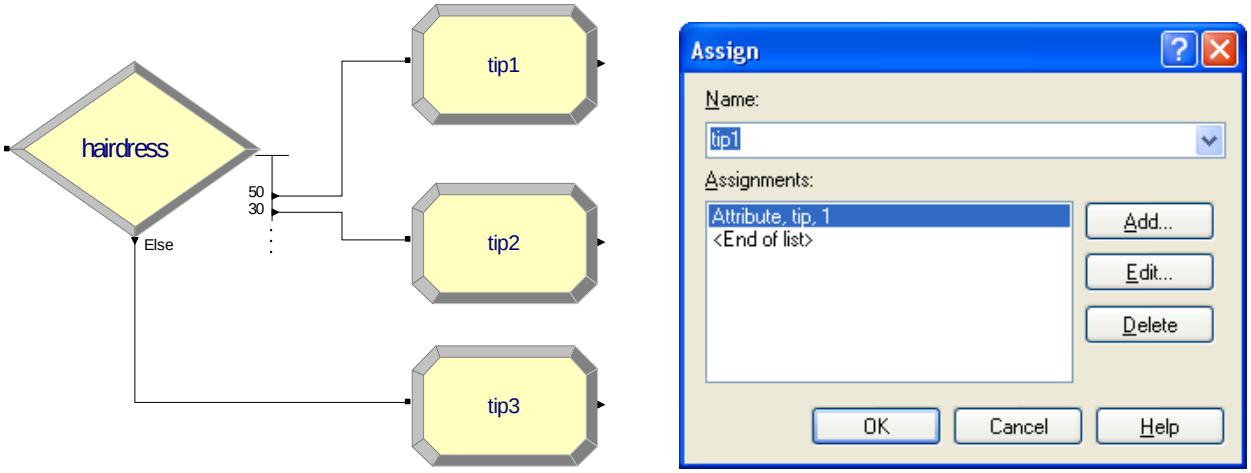
Double-click here to add a new row.

Initial Values

1	400
2	700
3	1200

Теперь нужно случайным образом выбрать для каждого поступающего транзакта-женщины тип желаемой прически. Будем хранить результат выбора в атрибуте *tip*.

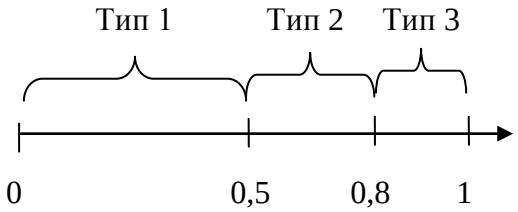
Выбор типа можно сделать двумя способами. Первый способ - с помощью модуля Decide и трех модулей Assign.



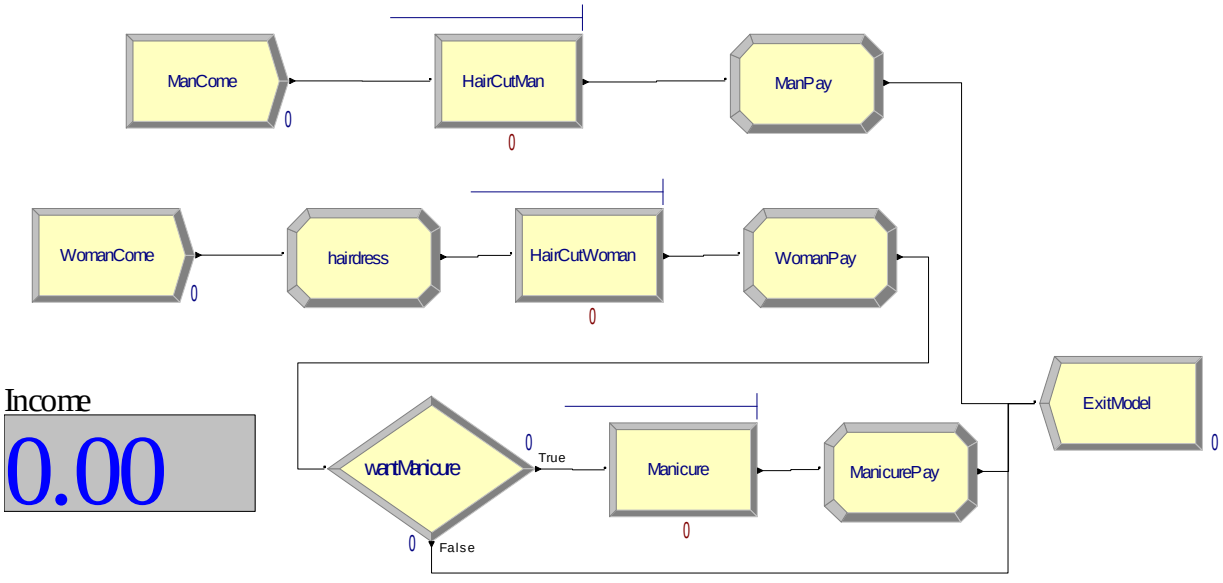
Второй вариант менее нагляден, но позволяет сэкономить число блоков модели. Воспользуемся функцией дискретного распределения. Заведем один модуль Assign, где атрибуту *tip* присвоим значение:

DISC(0.5, 1, 0.8, 2, 1.0, 3)

Эта функция требует указать для каждого возможного варианта накопленную вероятность и соответствующее значение.

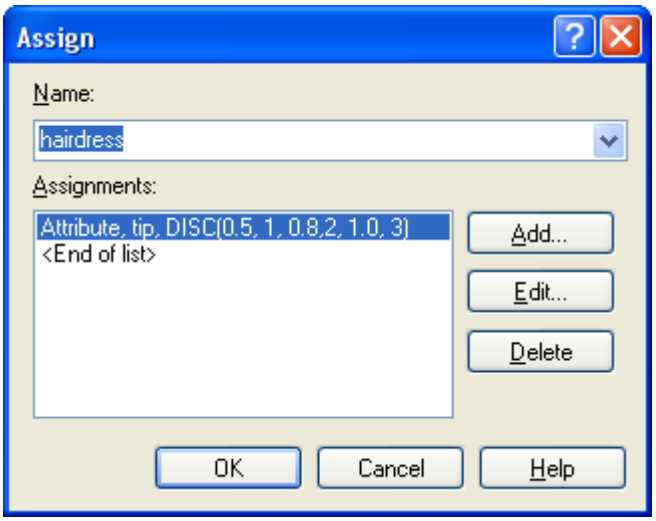


Исправленный граф модели:

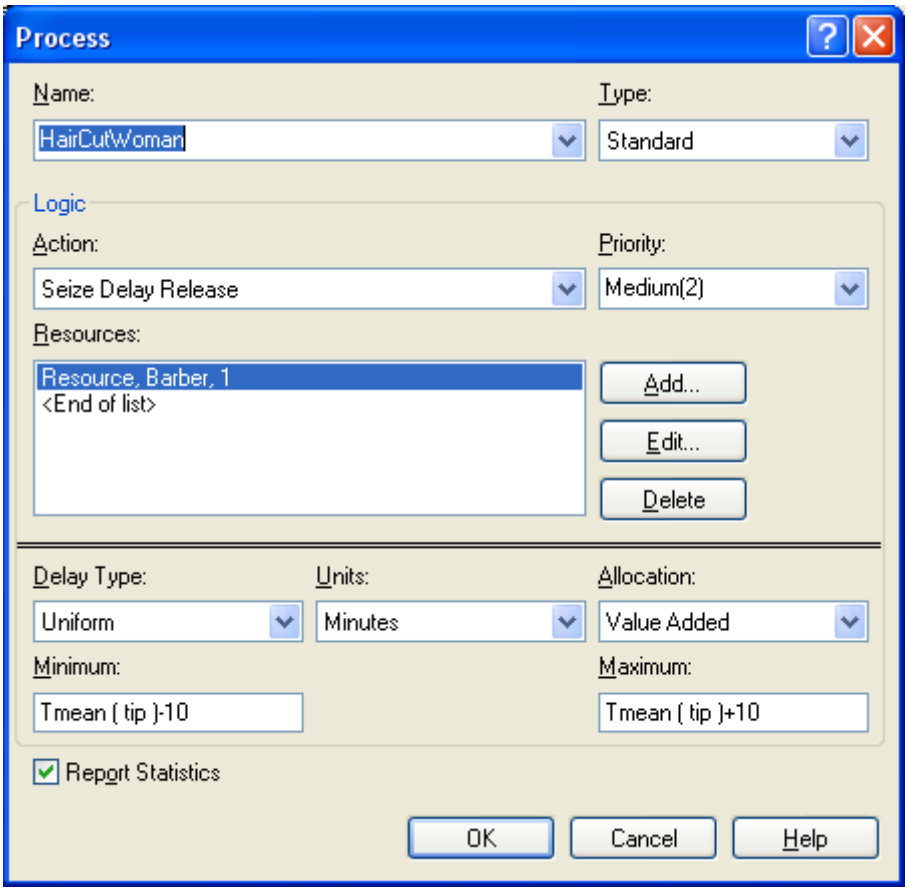


Обратим внимание на изменения в модели. Теперь за стрижку мужчин и женщин отвечают разные модули Process, хотя при этом и задействованы одни и те же ресурсы-парикмахеры.

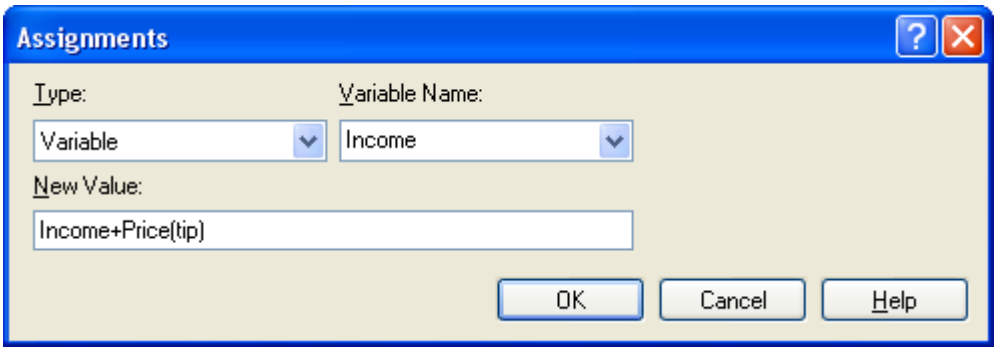
После генератора транзактов-женщин добавлен модуль типа Assign для выбора типа стрижки:



Новый модуль процесса HairCutWoman теперь использует значения массива для определения времени необходимого на стрижку. Подставляется то значение времени стрижки из массива *tmean*, которое соответствует типу стрижки, которую желает женщина.



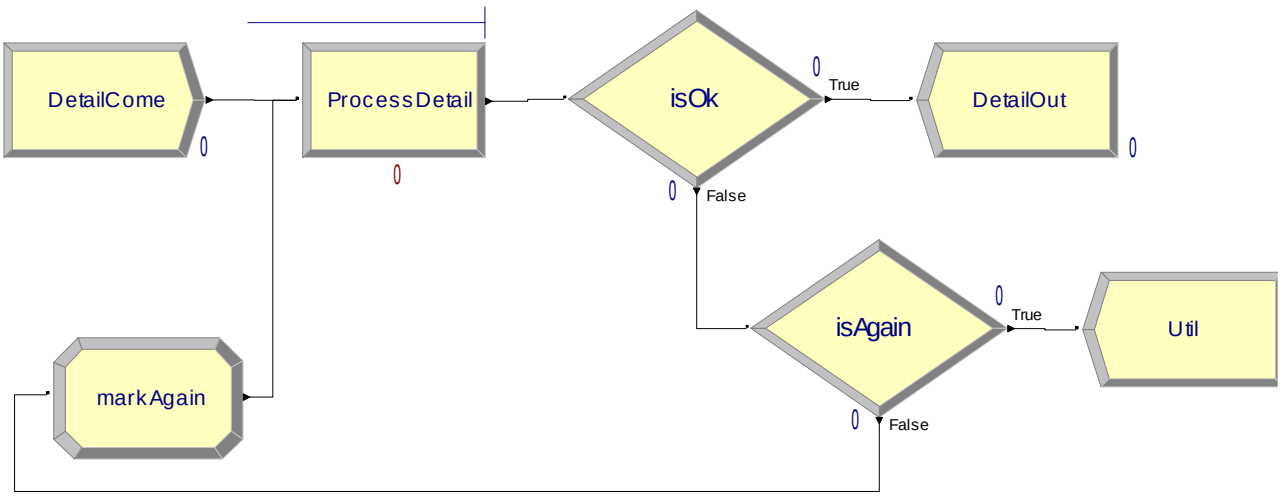
Будет исправлен и модуль типа Assign, где учитывается стоимость стрижки - WomanPay. Теперь цена зависит от типа стрижки (используется массив *price* из которого выбирается элемент с номером, соответсвующим типом стрижки – по атрибуту *tip*):



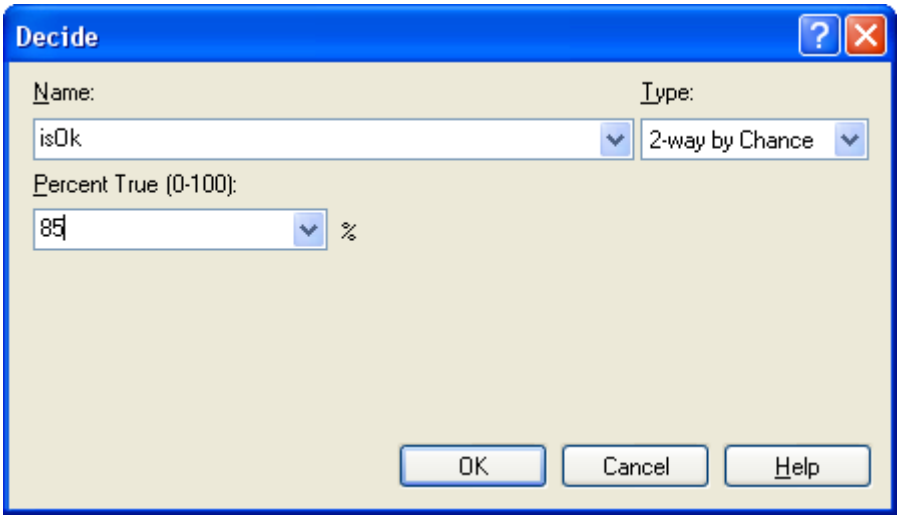
Пример 11. После обработки детали на станке с вероятностью 15% возникает брак, который может быть устранен повторной обработкой на том же станке. Однако если после повторной обработки деталь снова окажется бракованной, этот брак считается неустранимым и деталь отправляется в утиль.

Чтобы организовать подобную модель введем специальный атрибут *again*, означающий, что деталь пошла на второй круг. В граф модели добавим модуль Assign, в котором мы пометим деталь, установив этот атрибут и модуль Decide, где значение атрибута будет проверяться.

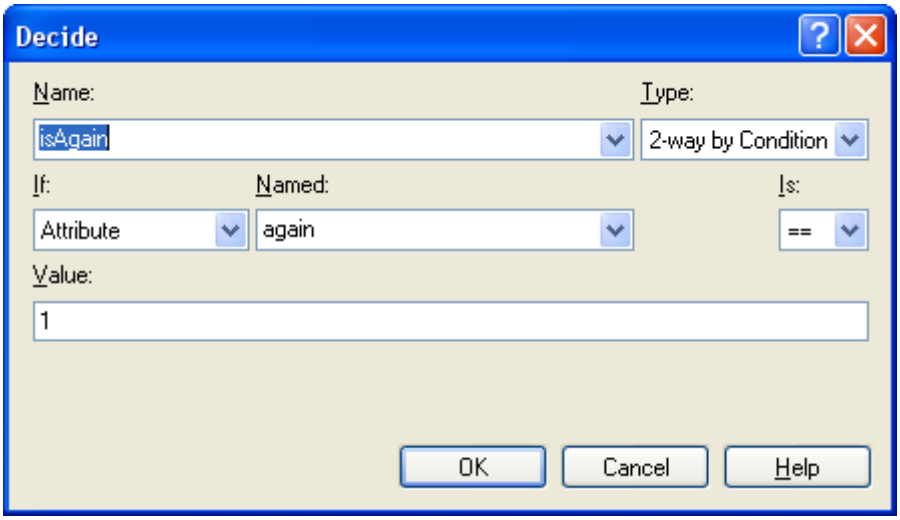
Граф представлен на рисунке.



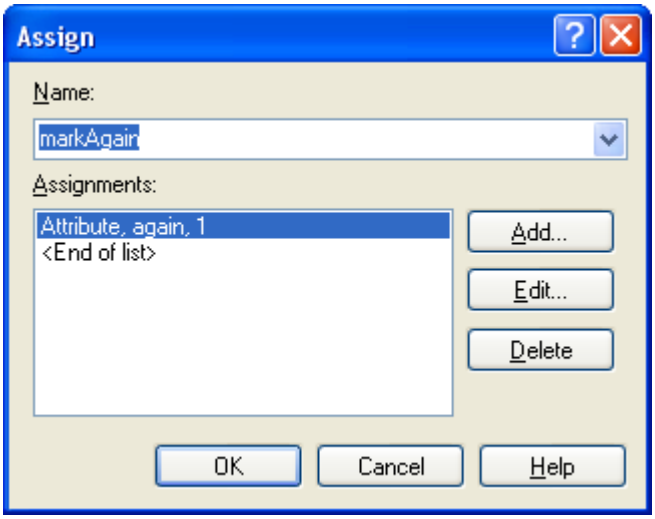
После модуля процесса ProcessDetail следует проверка с помощью модуля Decide с именем isOk, является ли деталь качественной. Выбран тип модуля «по вероятности».



Если деталь бракованная, осуществляется переход ко второму модулю Decide. Здесь установлен тип модуля «по условию». Выполняется проверка, не прошла уже деталь повторную обработку - если атрибут *again* равен единице. Этот атрибут играет роль метки. Если деталь прошла повторную обработку и по-прежнему является бракованной, то она направляется в утиль (модуль Dispose).



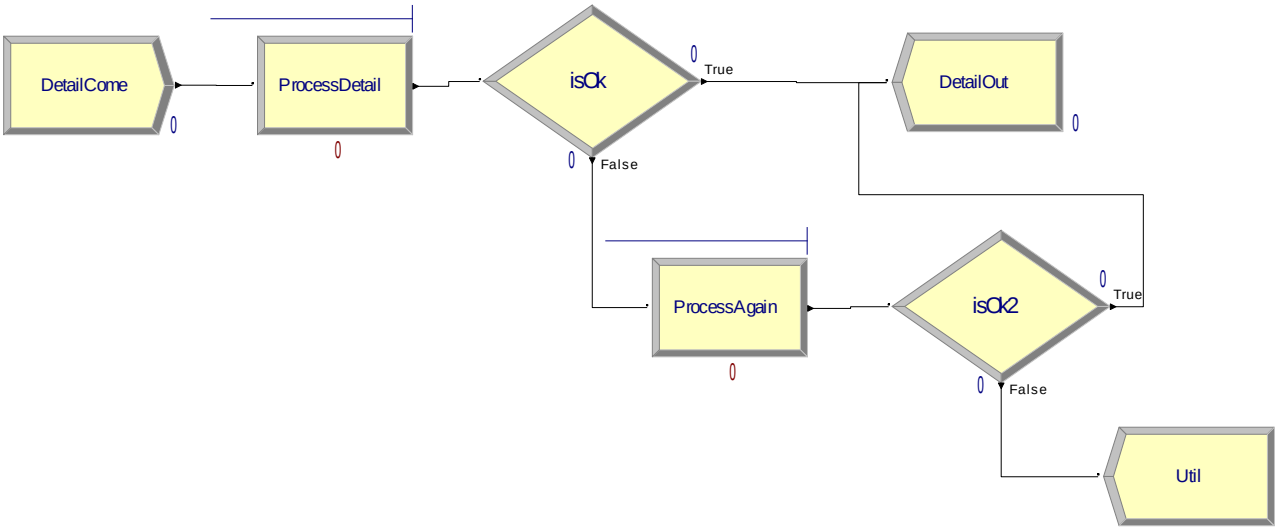
Если же бракованная деталь не помечена, то ее нужно направить на повторную обработку, но сначала на ней ставится метка с помощью модуля Assign. Атрибут *again* устанавливается в единицу. Далее деталь отправляется обратно в модуль ProcessDetail.



Обратим внимание, что атрибуты не нуждаются в предварительном объявлении. Инициация атрибута происходит в модуле assign при присвоении. Если значение атрибута не присвоено, то значением по умолчанию является 0. Проверка *again==1* даст результат *false*.

Рассмотренную в примере модель можно организовать и по-другому. Можно добавить для повторной обработки деталей соответствующий процесс и новый модуль контроля. Более того, подобная организация более корректна, ведь сейчас мы делаем два неявных предположения: повторная обработка бракованной детали занимает столько же времени, что и обработка новой детали; вероятность того, что деталь окажется бракованной, одинакова как после первичной обработки, так и после повторной обработки бракованной детали.

В модулях processDetail и processAgain можно использовать один и тот же ресурс.



Контрольные вопросы

1) В чем отличие модуля и блока?
2) Как в Arena можно реализовать абсолютный приоритет?
3) Что делает модуль Assign?
4) Чем является Entity type?
5) Как можно поменять изображение транзакта в ходе моделирования?
6) Можно ли в модуле Assign использовать случайные величины?
7) Как можно реализовать относительный приоритет?
8) Каким образом можно зафиксировать факт прохождения транзактом какой-то обработки?
9) Транзакты соответствуют документам на получение сумм денег. Как можно отразить факт наличия разных сумм у разных документов?
10) Как рассчитать общую сумму, полученную по всем документам?
11) Как можно совместно использовать модули Decide и Assign?
12) Если время обработки транзакта зависит от некоторого свойства этого транзакта (например, длина пакет данных или вес привезенного груза), как это нужно отразить в модуле Process?

5 Анимация и визуализация моделей

Анимация позволяет наглядно наблюдать процесс моделирования, визуально обнаруживать узкие места. Арена включает в себя достаточно мощную подсистему анимации «Cinema Animation».

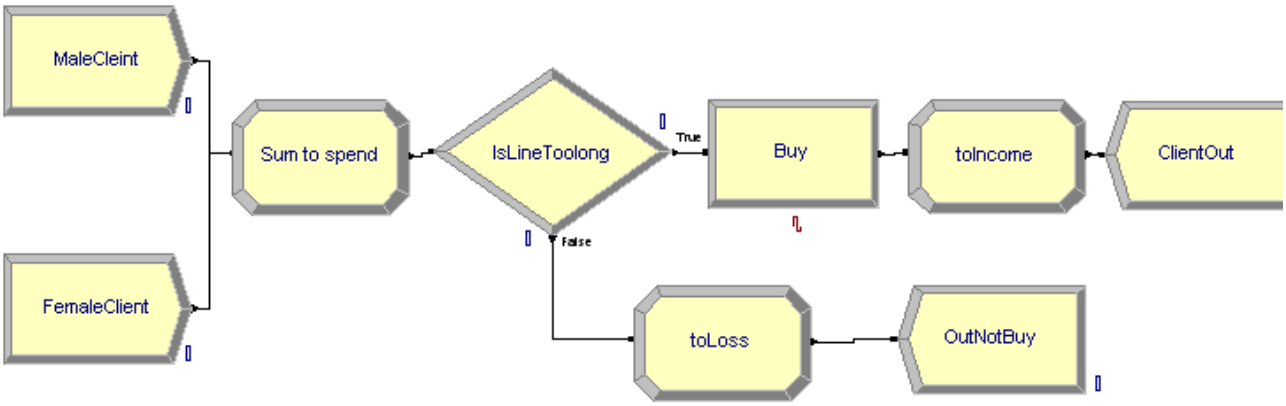
Для начала анимации от разработчика модели не требуется никаких действий. Как только граф модели построен и параметры заданы, перемещение транзактов будет отражаться на схеме, очереди будут заполняться, цифры выходов возле модулей будут отражать происходящее. Тем не менее, можно существенно расширить анимацию, визуализировать происходящее более детально.

Рассмотрим средства анимации для визуализации относительно простых моделей.

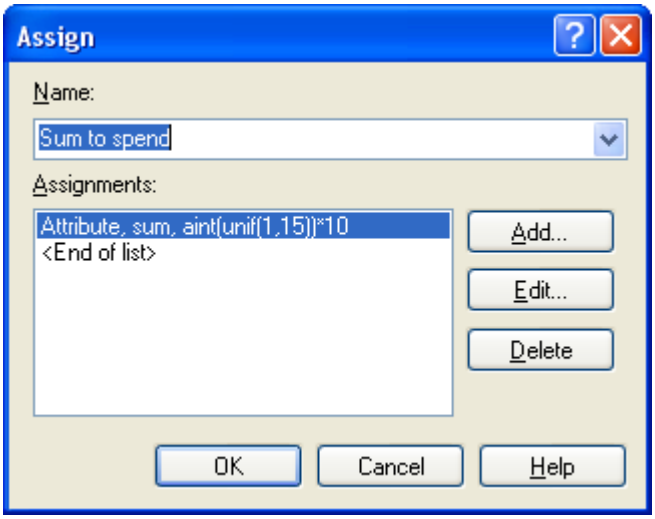
Сначала, разберем пример, который будем описывать средствами анимации.

Пример 12. В магазин приходят покупатели – мужчины и женщины. Каждый покупатель собирается потратить от 10 до 150 долларов. Покупателей обслуживает один продавец. Если покупатель обнаруживает очередь больше 5 человек, он уходит из магазина не совершив покупки.

Как видим, логика модели предельно проста и граф модели нуждается в минимальных пояснениях.



Первый модуль Assign устанавливает значение атрибута *sum*. Это сумма денег, которую планирует потратить покупатель.



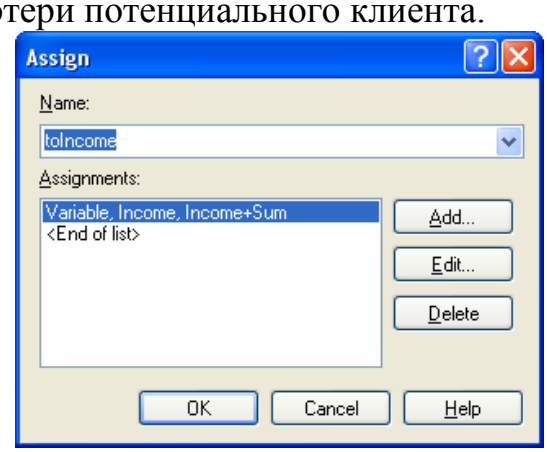
Функция *AINT()* производит округление числа до ближайшего целого, таким образом, получаем целое число, равномерно распределенное от 1 до 15 и, умножая на 10, получаем сумму соответствующую условиям задачи.

Модуль Decide проверяет длину очереди с помощью функции *NQ()*:

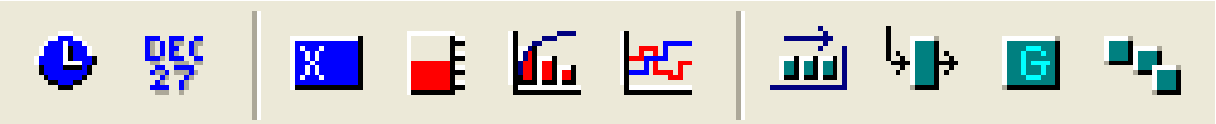
NQ(Buy.Queue)<5

Если в очереди меньше пяти клиентов, покупатель проходит к прилавку, иначе – уходит.

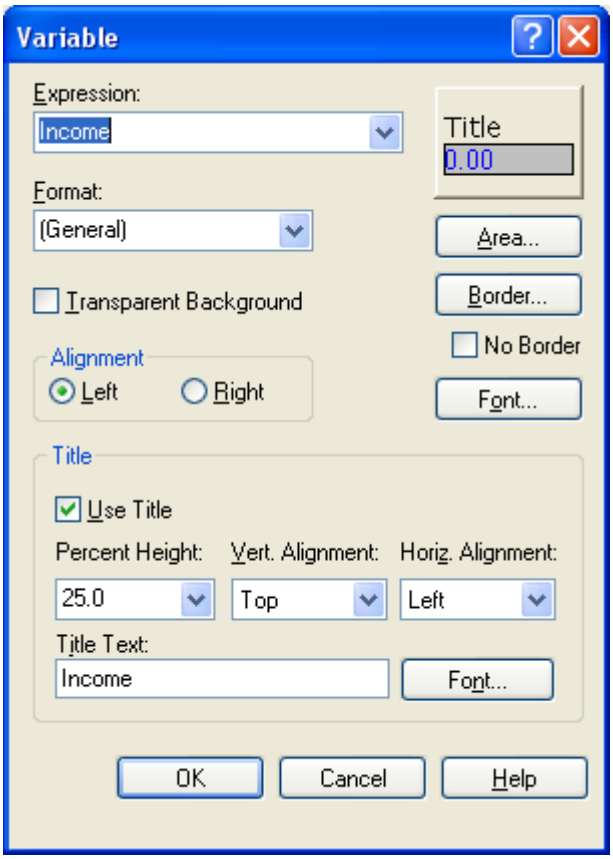
Остальные два модуля Assign призваны учесть прибыль и убытки. Для этого используются две переменные: *income* (выручка) и *loss* (потери). В модуле Assign с именем toIncome значение атрибута *sum* прибавляется к переменной *income*, чтобы учесть выручку от покупателя. В модуле toLoss значение этого атрибута прибавляется к переменной *loss*, чтобы показать убыток от потери потенциального клиента.



Теперь обратимся к средствам анимации, чтобы представить различные аспекты модели. Элементы анимации, показаны на специальной панели инструментов.

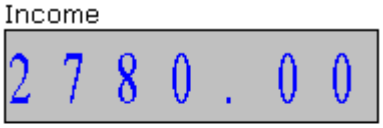


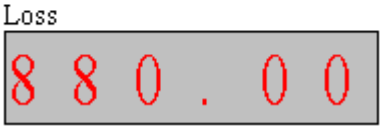
Прежде всего, воспользуемся средствами анимации переменных  и будем отражать текущие значения выручки и потерь:



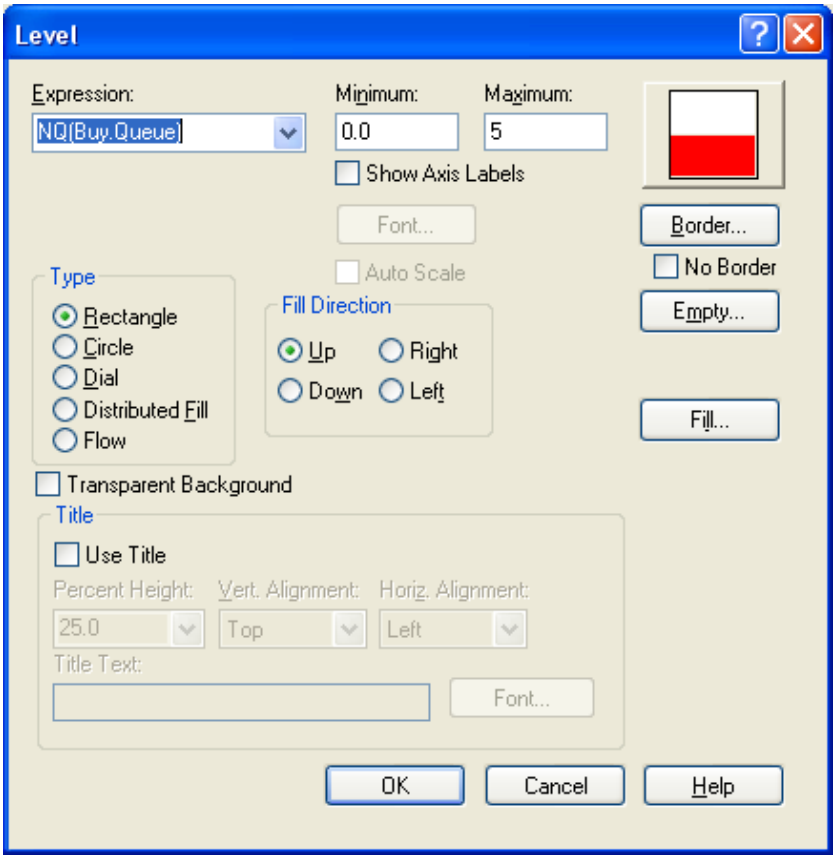
Главным здесь является поле Expression – выражение, значение которого отображается на модели. Мы устанавливаем здесь значение переменной *income*. Вообще же мы можем выводить любые выражения Arena, например, число элементов в очереди *NQ(…)*, число обслуженных покупателей (Female.NumberOut), загрузку ресурса и т.д.

Можно добавить для переменной название, установив флажок Title и указав это название. С помощью кнопки Font можно выбирать цвет – выручку (*income*) будем отражать синим, а потери (*loss*) – красным.



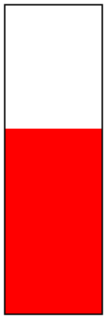


В нашей модели ограниченный объем очереди, максимум составляет 5 человек. Попробуем отразить это визуально. Для подобных целей [18] используют специальные визуальный элемент «Level» (уровень)  .



Мы определили выражение – число покупателей в очереди. Задали минимальное (Minimum) и максимальное (Maximum) значения.

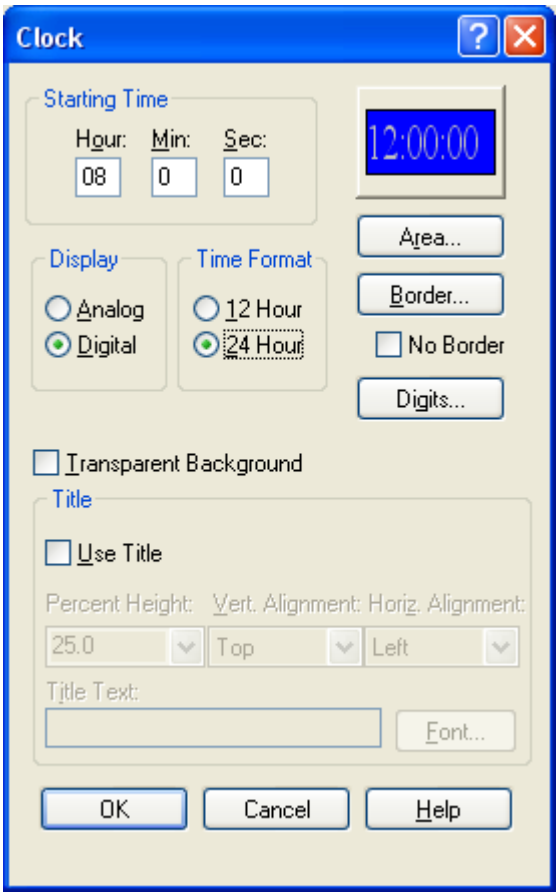
Также можно поменять цвет, направление для заполнения (например, слева-направо), форму графического элемента.



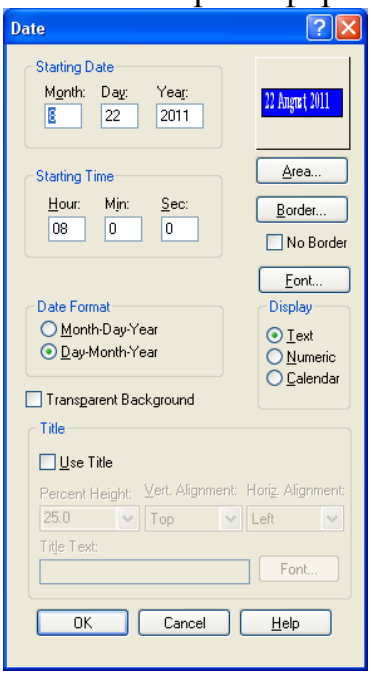
Теперь будем отслеживать модельное время. Для этого используем визуальные средства часы и календарь 

.

[18] Другая аналогичная задача – отразить процент занятых единиц ресурса.

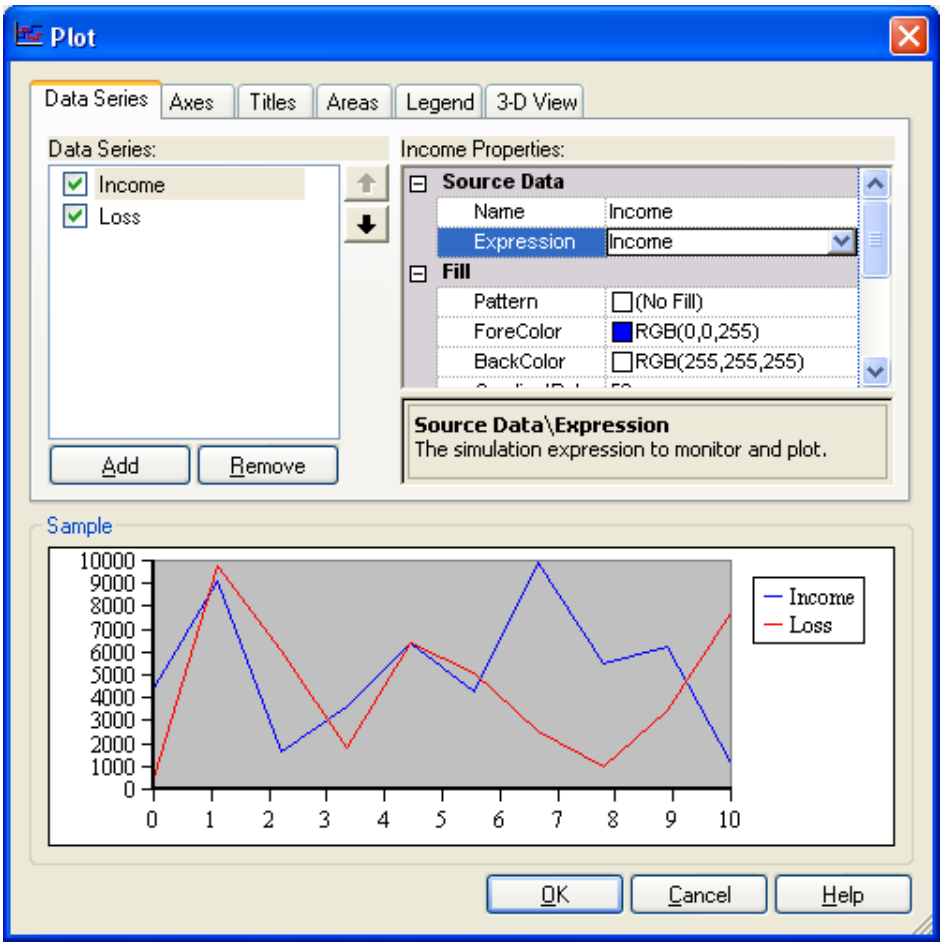


Указываем начальное время (мы считаем, что начало рабочего дня – 8:00). Выбираем формат часов – аналоговый (часы со стрелкой), либо цифровой, а также выбираем формат – двенадцати- или двадцатичетырехчасовой.



Для календаря выбираем начальную дату, время и правильный формат даты.

Следующий элемент анимации – графики  . Создадим график для отслеживания накопления выручки и потерь.



Необходимо определить список серий, в данном случае их две – для прибыли и убытков. Для каждой серии зададим:

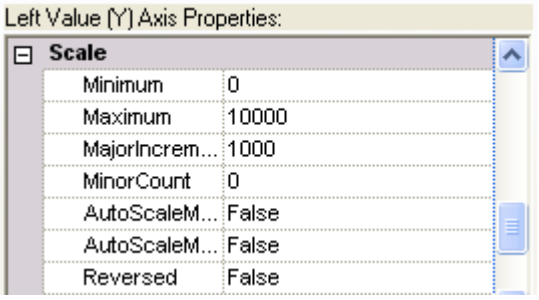
- имя серии (name);

- выражение (expression), которое отражается на графике. В нашем случае это значение переменных *Income* и *Loss*, но здесь можно ввести любое выражение Arena;

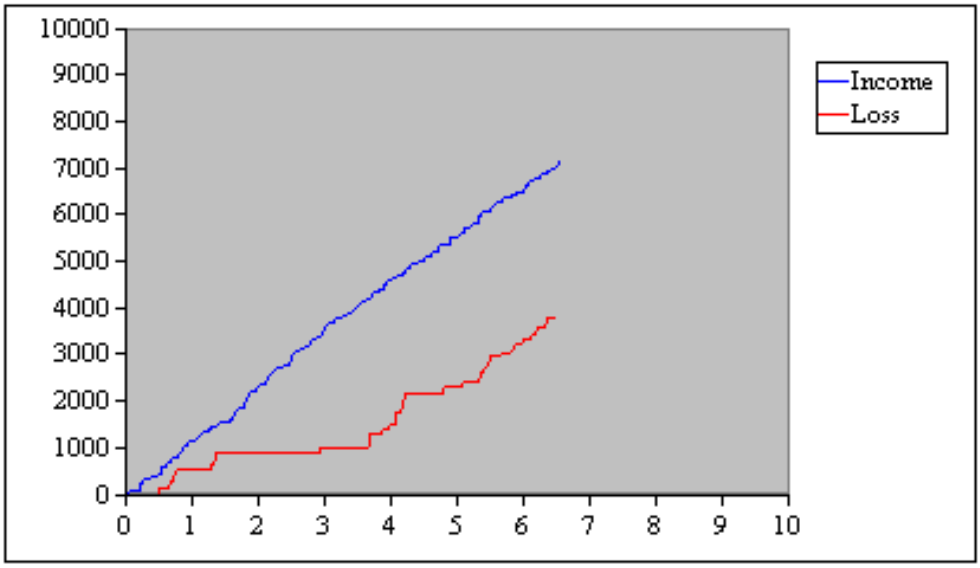
- цвета графика и надписи.

На вкладки Axes можно калибровать оси графика. В частности можно задать минимум и максимум для оси X в базовых единицах времени.

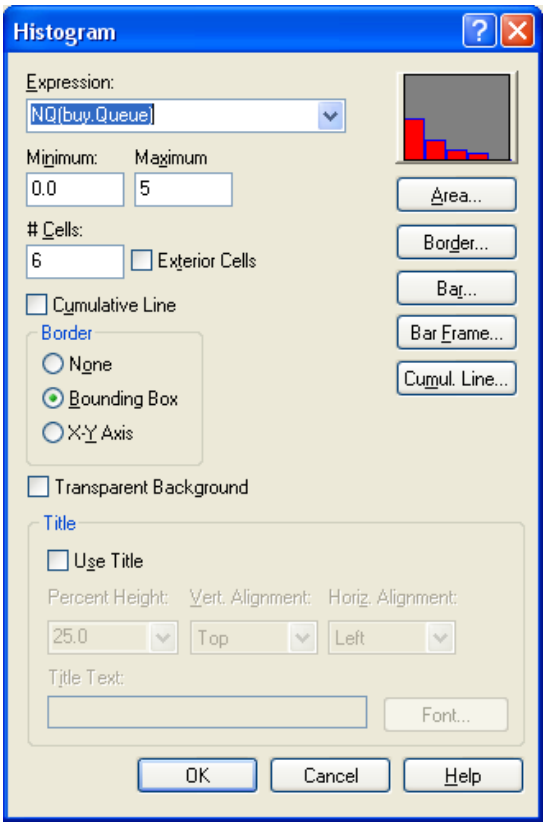
Также для оси Y можно указать минимум, максимум и шаг. Мы задали значения от 0 до 10000 с шагом 1000, соответствующим образом отражается ось графика. Свойства AutoScaleMinimum и AutoScaleMaximum мы сбросили в false - теперь график не регулирует эти значенияавтоматически, а принимает введенные нами границы.



Результат при моделировании виден ниже.

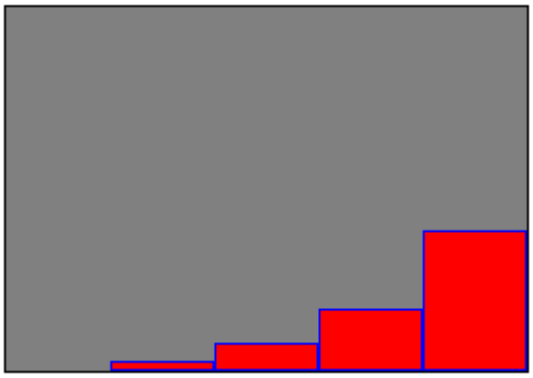


Следующий элемент – гистограмма .

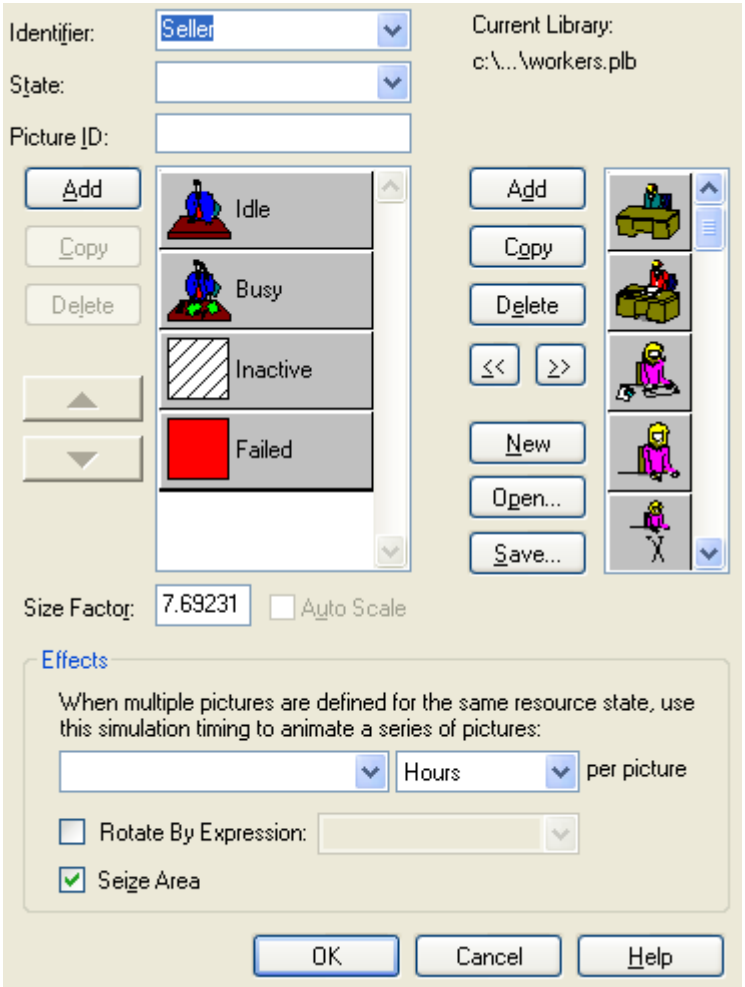


Мы строим гистограмму числа элементов в очереди, поэтому задаем выражение *NQ(Buy.Queue)*. Далее надлежит задать минимальное и максимальное значения , а также число интервалов.

Результат показан ниже:



Приступим теперь к анимации ресурса с помощью соответствующего элемента анимации Resource  . В появившемся окне выбираем идентификатор ресурса, в данном случае Seller (продавец).



Здесь мы имеем возможность выбрать несколько картинок для ресурса – для каждого из его состояний.

Правый список – графическая библиотека. Можно подгрузить другую библиотеку (кнопка Open) или создать свою (кнопка New). Можно копировать рисунки (кнопка Copy), создавать собственные (кнопка Add), удалять (кнопка Delete). Можно редактировать рисунок, дважды нажав на нем кнопкой мыши.

Слева на рисунке показан список состояний ресурса. С помощью кнопок << можно установить для каждого состояния картинку (в принципе, для одного состояния можно сделать несколько картинок и указать скорость их смены, в этом случае, например, когда станок работает, можно показать, как летят искры).

Мы выберем свободного сотрудника для состояния IDLE и сотрудника с загруженным столом для состояния BUSY.

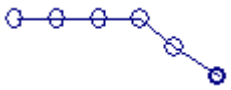
Установим флажок «Sieze area» – так мы сможем видеть, какой транзакт в данный момент обслуживается ресурсом.

Нажав OK, размещаем анимацию ресурса на полотне модели. Кружок с пунктиром – это и есть «Sieze area», место для изображения текущего транзакта.

Далее приступим к анимации очереди.

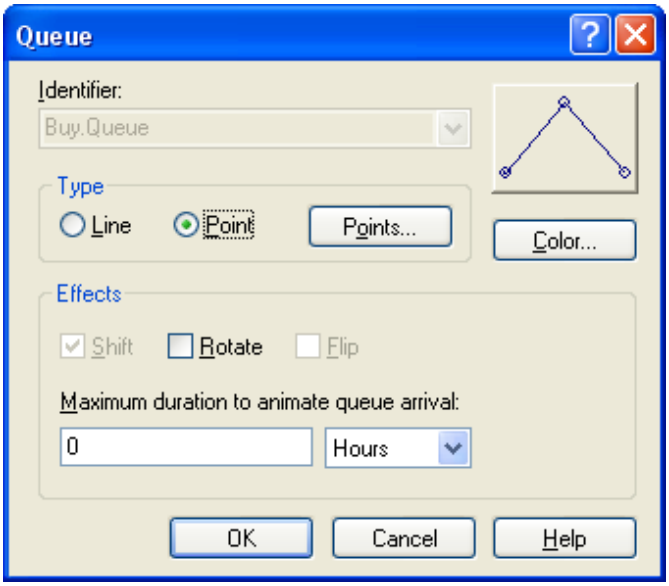
Очередь может отражаться одним из двух способов:

- линейный 

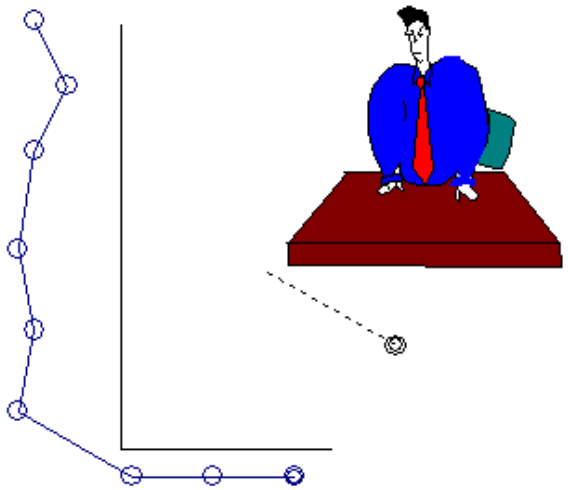
- точечный 

Во втором случае, можно размещать позиции для ожидающих транзактов так, как необходимо для рисунка.

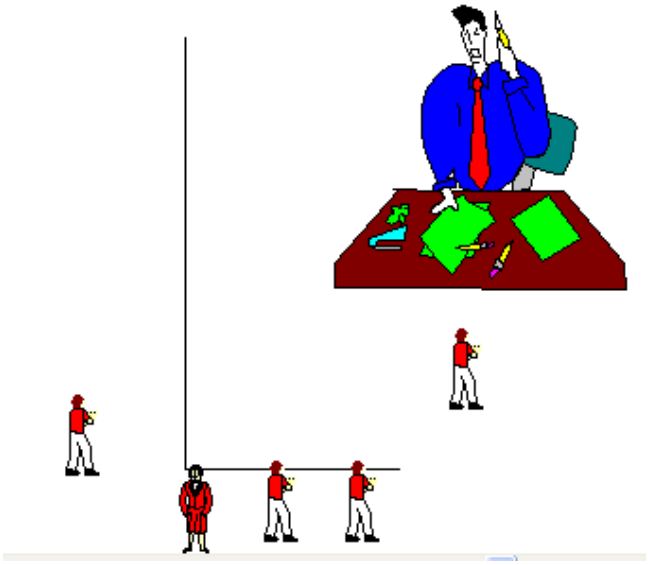
Нарисуем с помощью линий прилавок вокруг продавца. Теперь заставим очередь огибать прилавок. Перетащим изображение очереди от модуля процесса к продавцу. Далее нажав на очереди, выведем окно свойств и в нем изменим тип на точечный (point).



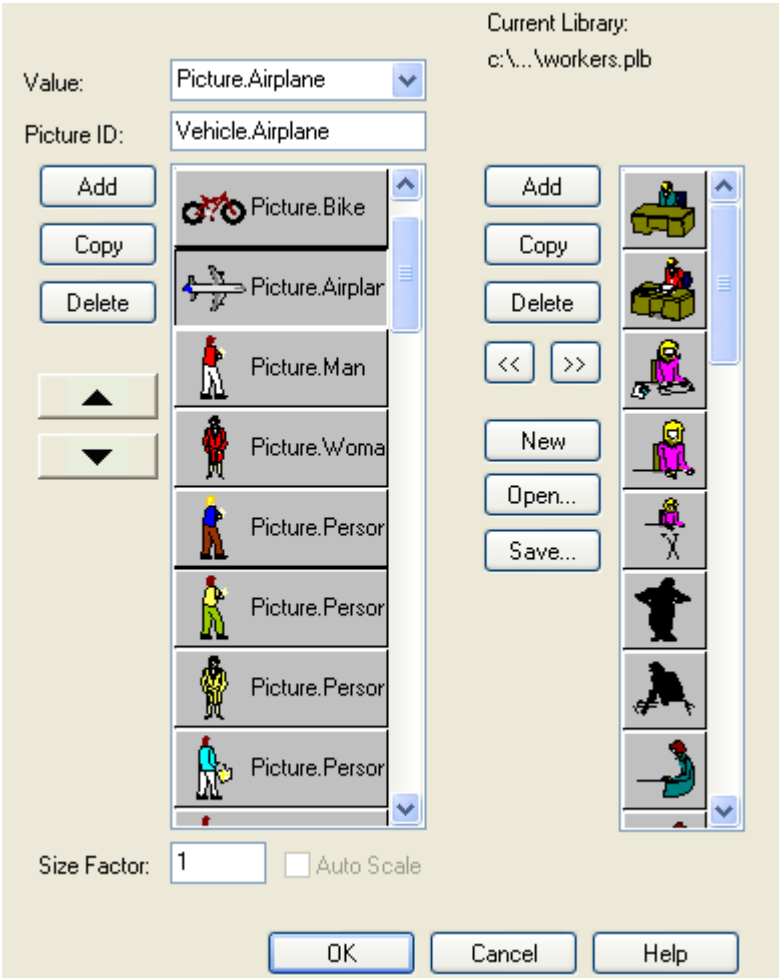
С помощью кнопки Points добавим нужное число точек. Далее нажимаем OK и перетаскиваем с использованием левой кнопкой мыши отдельные точки так, как нам нужно в соответствии со схемой.



При анимации модели результат выглядеть следующим образом:



Рассмотрим теперь анимацию транзактов. Изображение транзакта выбирается при его создании в невизуальном модуле Entity. Можно, однако, разработать и собственные изображения. Для этого имеется пункт меню Edit – Entity Pictures. Здесь можно подгружать графические библиотеки, а также рисовать свои изображения или редактировать существующие.



Слева показан список картинок. Каждая картинка описывается идентификатором, например Picture.airplan. Именно его мы устанавливаем в атрибуте *entity.picture*.

Можно задать для одного идентификатора множество картинок – тогда для соответствующих транзактов картинки будут выбираться случайным образом. Пример – идентификатор транзакта Picture.Person, которому соответствует множество различных изображений мужчин и женщин.

6 Использование пулов (set) ресурсов

Arena предоставляет еще одну возможность работы с ресурсами – это пул ресурсов (set). Используется также термин «множество ресурсов». Цель применения множеств состоит в повышении гибкости модели: исследователь создает отдельные ресурсы, затем объединяет их в множества, а затем в блоке Process указывает не конкретный ресурс, а множество. Например, пул «Водители» может включать ресурсы «Иванов», «Петров» и «Зайцев». При обработке транзакта (задания не перевозку) не важно, кто именно из водителей будет производить рейс. При поступлении транзакта будет выбран свободный водитель из пула.

В результате повышается детализация и гибкость – ведь каждый ресурс моделируется отдельно, ресурсы не совсем одинаковы, есть возможность задать для каждого свои характеристики (влияющие на стоимость выполнения работ или на их продолжительность). Например, у фирмы может быть несколько станков с разной производительностью. Можно собирать статистику по каждому ресурсу.

Обратим внимание, что один и тот же ресурс может входить в несколько пулов.

Рассмотрим пример. В качестве ресурсов выступают сотрудники:

Resource - Basic Process

	Name	Type	Capacity
1	Pasha	Fixed Capacity	1
2	Masha	Fixed Capacity	1
3	Igor	Fixed Capacity	1
4	Lena	Fixed Capacity	1
5	Ivan	Fixed Capacity	1
6	Vika	Fixed Capacity	1
7	Semen	Fixed Capacity	1

Double-click here to add a new row.

У нас также есть несколько пулов:

Set - Basic Process

	Name	Type	Members
1	Managers	Resource	5 rows
2	Drivers	Resource	1 rows
3	SeniorManagers	Resource	2 rows
4	Chief	Resource	2 rows

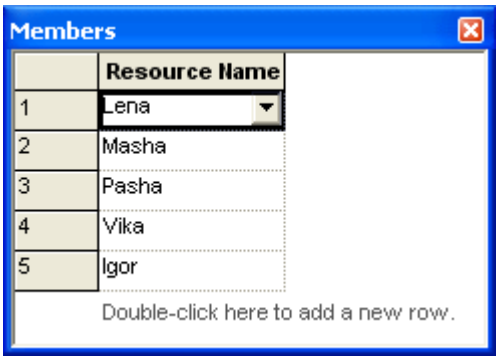
Double-click here to add a new row.

Как видим, здесь представлены водители, менеджеры, старшие менеджеры (выполняют какие-то определенные работы, требующие высокой квалификации), руководители (имеющие свой особый круг обязанностей, на-

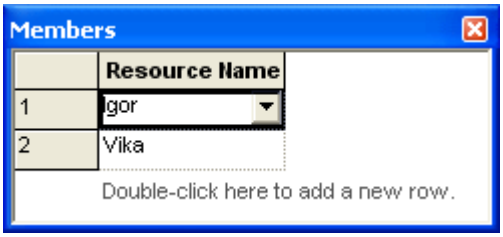
пример: принимать участие в совещаниях, планировать работу подразделений и т.д.).

Нажав кнопку в строке таблицы можно открыть список элементов пула и выбрать ресурсы из имеющихся, чтобы включить их в этот пул.

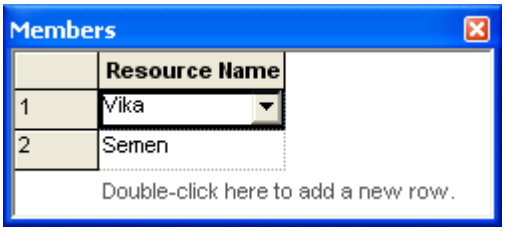
Пул менеджеров:



Пул старших менеджеров:

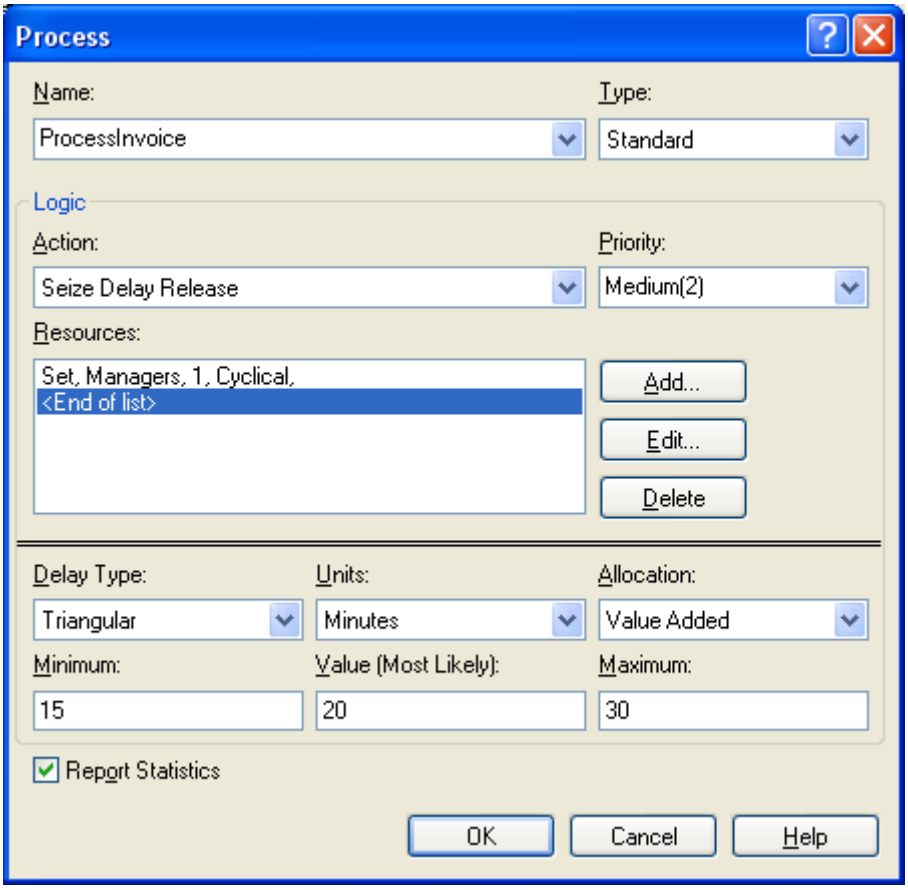


Пул руководителей:

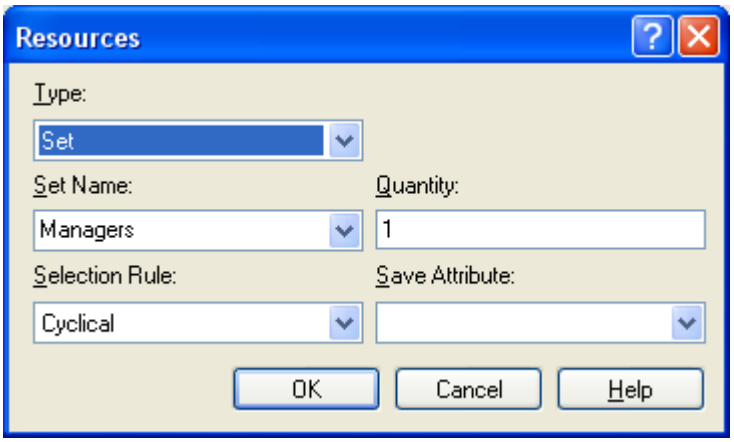


Обратим внимание: Вика и Игорь входят в пулы «менеджеры» и «старшие менеджеры» - это значит, они могут выполнять сложные работы, но также при необходимости выполнят и стандартные операции менеджеров. При этом Вика является еще и руководителем, если будет проводиться совещание, Вика будет привлечена к этому совещанию (скорее всего модули Process, связанные с совещанием будут иметь высший приоритет).

Рассмотрим теперь операцию обработки поступившей расходной накладной, представленную модулем Process с задействованными ресурсами. Параметры модуля приведены на рисунке.



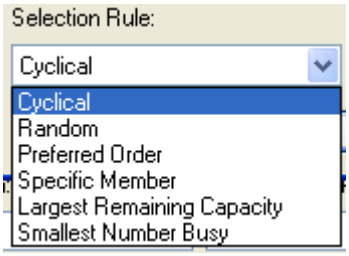
Как видим, в списке ресурсов вместо конкретного ресурса указан пул – обработку накладной может выполнить любой из свободных менеджеров. Окно подключения ресурсов показано на рисунке ниже.



В поле Type этого окна указываем «Resource», если хотим точно указать конкретный ресурс или «Set», если хотим использовать множество, пул ресурсов.

В данном случае выбран тип «Set», задано имя пула («Managers») и указано, что для операции нужен один менеджер.

Правило выбора «Selection Rule» определяет, какой именно ресурс из свободных в данный момент будет выбран.



Возможны следующие варианты выбора:

- *последовательный выбор* (Cyclical). Члены пула будут выбираться по порядку, при очередном обращении будет выбираться следующий свободный ресурс. Например, если первую операцию выполняла менеджер Лена, следующую будет выполнять менеджер Маша и т.д. Это наиболее распространенный вариант, обеспечивающий равномерную загрузку ресурсов;

- *случайный ресурс* (Random);

- *в порядке предпочтения* (Preferred Order). Будет выбран самый верхний в списке из свободных ресурсов пула. Этот вариант встречается, если: ресурсы имеют разную производительность и желательно использовать более быстрый; желательно задействовать более опытного работника; ресурсы имеют разную стоимость машино-часа и желательно задействовать более дешевый; некоторые ресурсы могут требоваться в другом месте и отвлекать их стоит только в случае необходимости – в последнюю очередь;

- *определенный член пула* (Specific member). При выборе этой опции этом появляется поле, где следует указать номер ресурса в пуле. Как правило, этот вариант используется в модуле Process типа Delay-Release (чтобы отпустить захваченный ранее и удерживаемый транзактом ресурс из пула);

- *ресурс с наибольшим числом свободных единиц* (Largest remaining capacity);

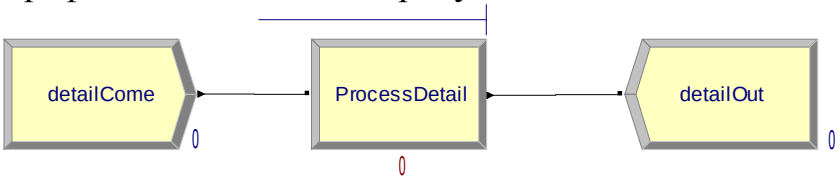
- *наименее загруженный* (Smallest number busy).

Два последних варианта используются, если ресурсы пула имеют мощность большую единицы.

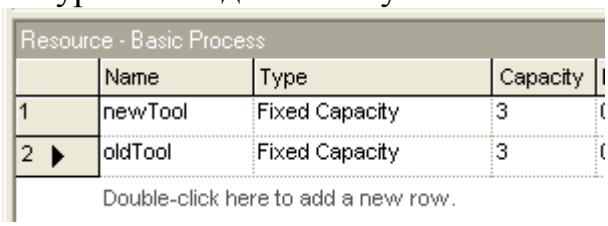
В поле Save Attribute указывается атрибут транзакта, в который будет записан номер выбранного ресурса из пула.

Пример 13. На предприятии работают 3 новых и 3 старых станка. Старый станок тратит на обработку детали в среднем 30 минут при среднеквадратическом отклонении 3. Новый тратит 20 минут при том же среднеквадратичном отклонении. Если новый станок свободен, обработку следует проводить на нем. Детали поступают раз в 7 минут, закон распределения экспоненциальный. Построить модель, оценить загрузку оборудования.

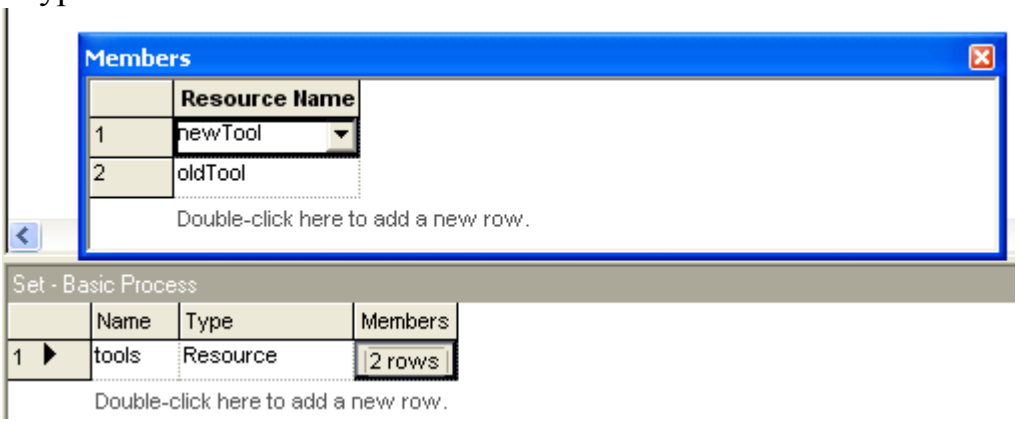
Граф модели показан на рисунке.



Ресурсами модели выступают станки – старый и новый:

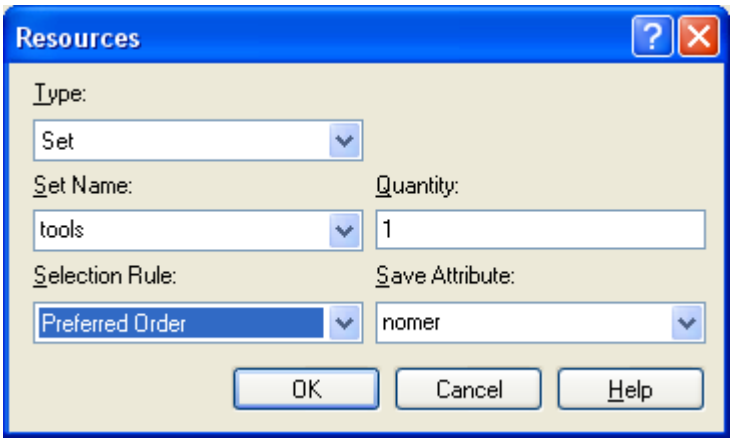


С помощью невизуального модуля Set создаем пул Tools и включаем в него ресурсы:



Обратим внимание, на порядок ресурсов в списке – новый станок идет первым, поскольку именно его мы хотим выбирать, если это возможно.

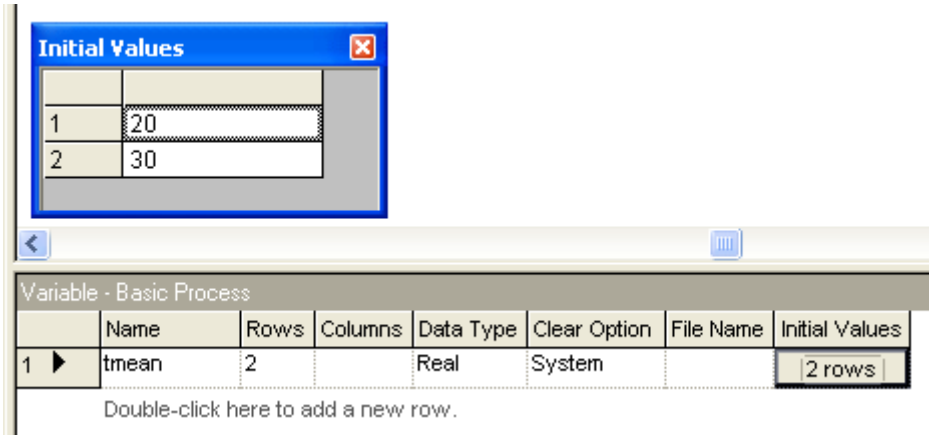
Для модуля Process в окне редактирования ресурса указываем, что нам нужен не конкретный ресурс, а пул (то есть в поле Type выбираем значение «Set»):



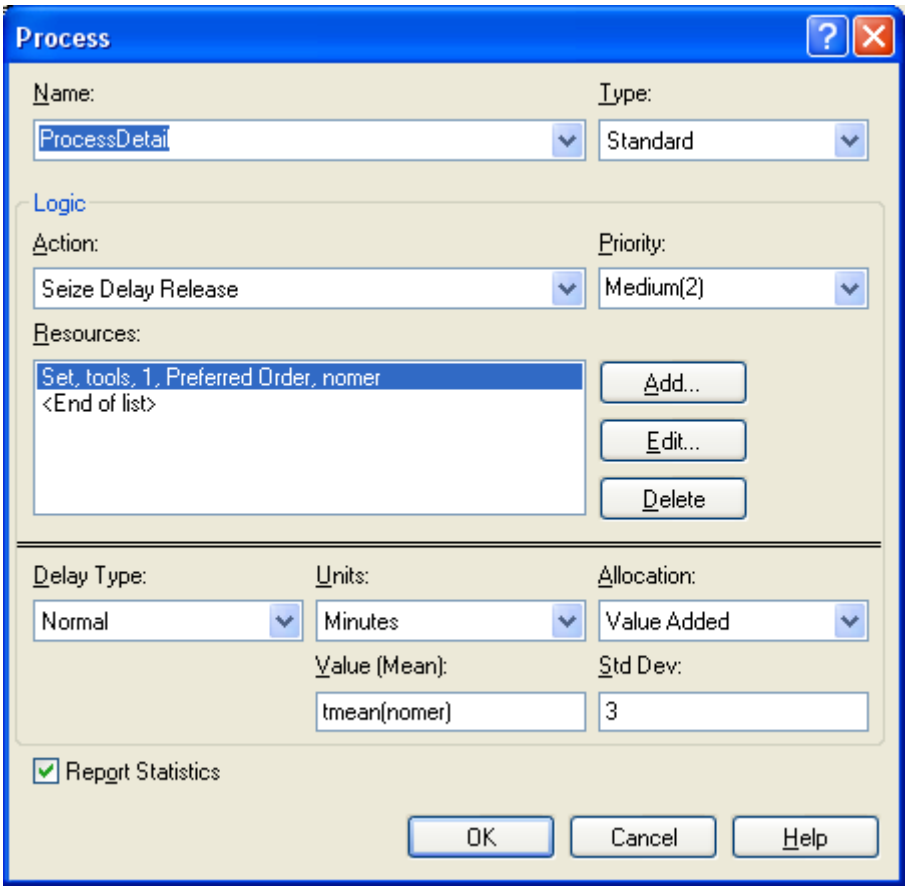
Выбираем название пула (Tools). Указываем Selection Rule (правило выбора) «Preferred Order». Теперь, если один из новых станков свободен, деталь будет обрабатываться на нем.

В поле Save Attribute указываем атрибут (мы назвали его *nomer*). В атрибут *nomer* теперь будет занесен номер типа станка, который будет выбран из пула для обработки детали. Зная номер выбранного ресурса, мы можем использовать этот номер в модели.

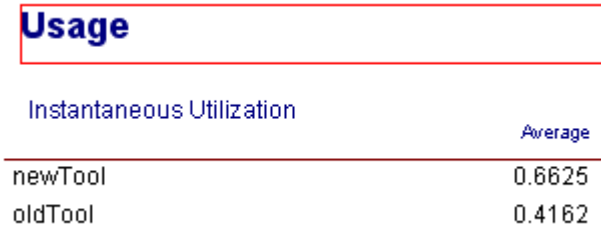
Заведем переменную-массив *tmean*, где будем указывать ожидаемое время обработки детали.



Элемент массива *tmean* с индексом *nomer* будем указывать в модуле Process при определении времени обработки детали (тогда для нового станка ожидаемое время будет 20 минут, для старого - 30):



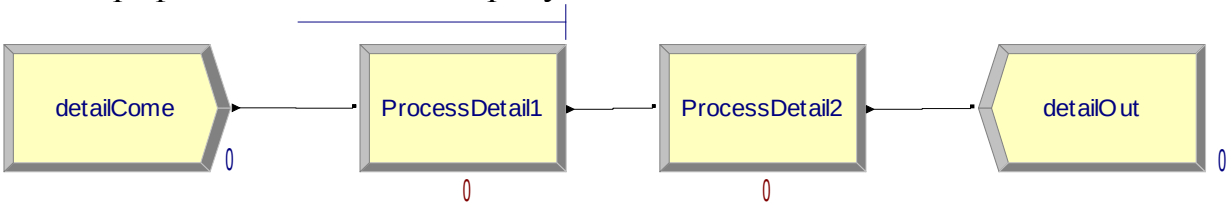
В отчете о загрузке ресурсов, полученном в результате моделирования, мы видим, что новые станки задействованы чаще.



Пример 14. Несколько усложним пример. Допустим, на выбранном станке нужно выполнить не одну, а две последовательных операции. Вторая операция, как на старом, так и на новом станке занимает 4+-2 минуты, закон распределения равномерный. Обратим внимание, что рабочий проводит вторую операцию сразу же после первой, не снимая деталь со станка.

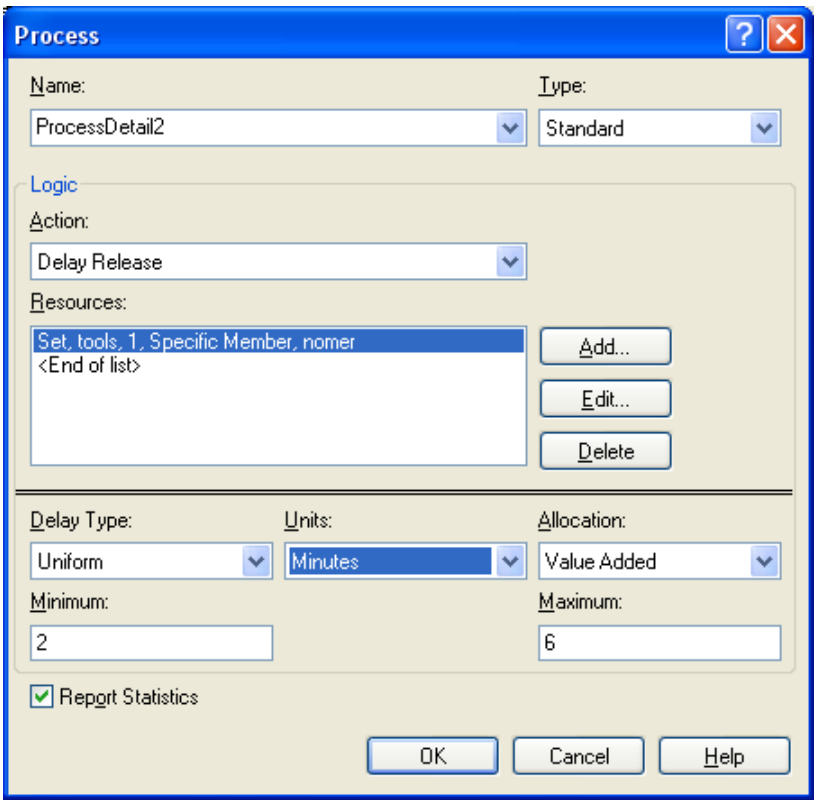
Таким образом, имеем многошаговую модель, и транзакт-деталь должен захватить ресурс, удерживать его, а затем отпустить уже в другом модуле.

Граф модели показан на рисунке.

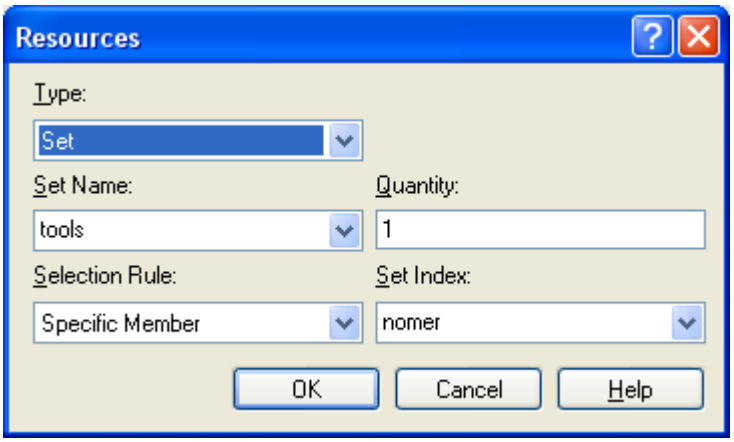


Единственное изменение в первом модуле Process – его тип изменился на Sieze-Delay. То есть теперь занятый ресурс не отпускается, а удерживается транзактом.

Второй модуль Process с именем ProcessDeatail2 соответствует второй выполняемой операции. Эта операция делается на том же самом станке, что и первая (на удерживаемом транзактом ресурсе). Соответственно, тип модуля Delay-Release.



Возникает вопрос, как указать, какой именно ресурс нужно отпускать? Номер ресурса мы сохраняли в атрибуте транзакта с названием *nomer*. Именно этим атрибутом мы и воспользовались.



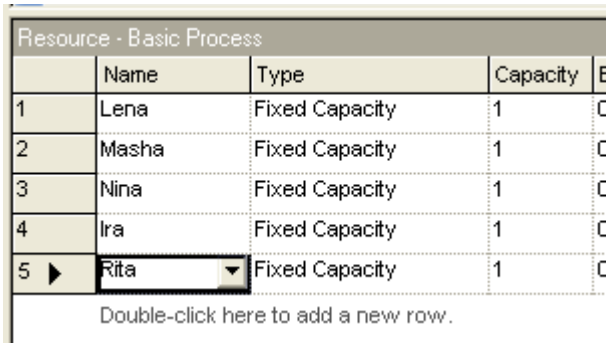
Как видим, здесь мы также выбираем тип Set (пул ресурсов). Указываем имя пула ресурсов (tools), говорим, что нужно использовать и отпустить одну единицу ресурса.

Но в качестве правила выбора (Selection Rule) указываем «Speciefic Member» (конкретный номер) и указываем индекс (Set Index), а в качестве индекса называем атрибут *nomer*. Ведь именно в этом атрибуте транзакта-детали сохранен номер станка, который выбран для обслуживания.

Пример 15. Вернемся к примеру с парикмахерской (**Пример 1, Пример 6, Пример 9, Пример 10**). В прошлых примерах предполагалось, что любой парикмахер может выполнить любую стрижку. Будем теперь отслеживать их работу конкретных парикмахеров имеющих определенные специальности.

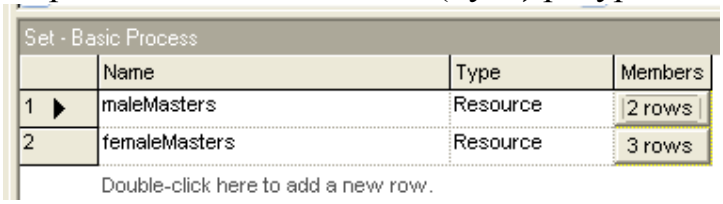
В парикмахерской работают:
- женские мастера: Ира, Нина;
- мужской мастер: Лена;
- мастер-универсал: Маша (может стричь и мужчин и женщин);
-специалист по маникюру: Рита [19].

Структура графа модели не изменится по сравнению с **примером 10**.
На рисунке показан список ресурсов.

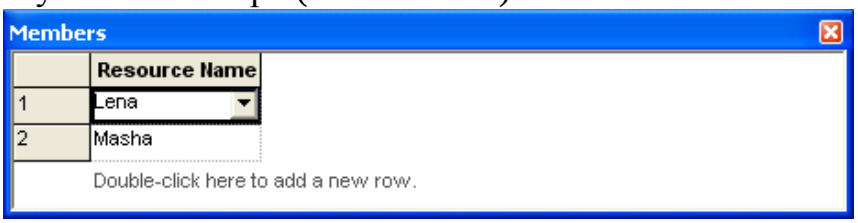


[19] Модель можно усложнять дальше, например, можно ввести новые услуги, можно указать группу элитных парикмахеров, которые могут делать эксклюзивную прическу и т.д. Сейчас мы этого делать не станем.

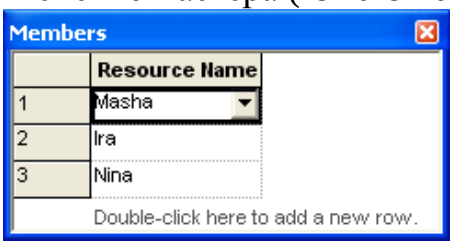
Определим два множества (пула) ресурсов:



Мужские мастера (maleMasters):

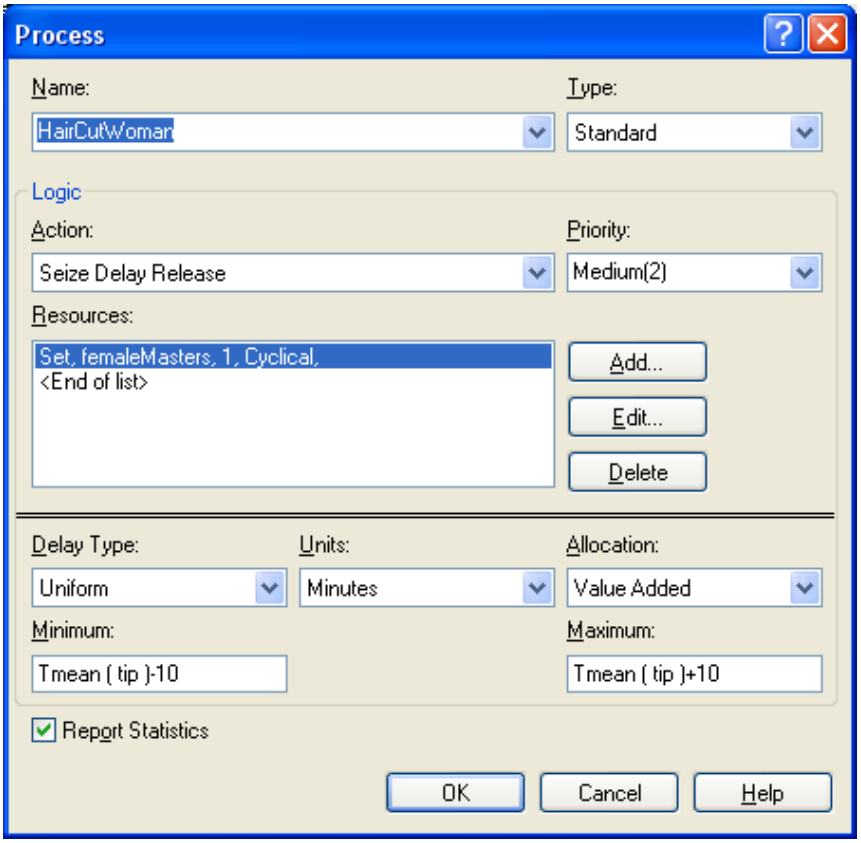


Женские мастера (femaleMasters):

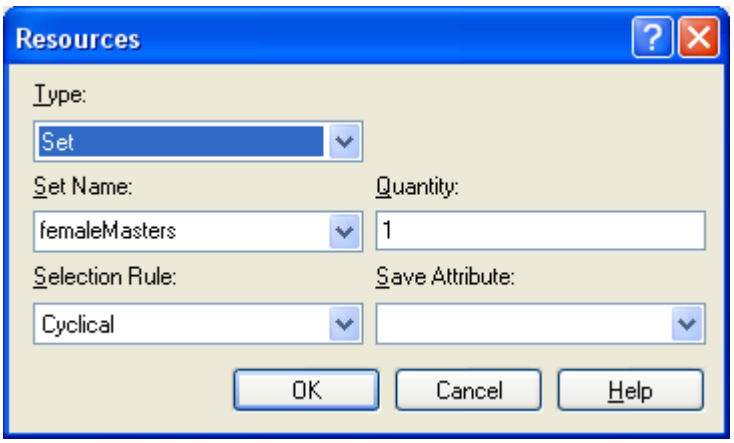


Обратим внимание, Маша, как универсал, входит в оба пула.

Модуль Process, отвечающий за маникюр, претерпел одно изменение – теперь мы явно указываем имя ресурса Rita (Рита является мастером-маникюра, поскольку она одна, мы не стали оформлять соответствующий пул, а указали имя ресурса явно).



В модуле Process для женской стрижки HairCutWoman теперь указываем не название ресурса, а пул ресурсов – femaleMasters.



Посмотрим изменения в отчете:

Wait Time

	Average
Man	15.2618
woman	7.4207

Из фрагмента отчета видно, что мужчинам приходится дольше ждать в очереди.

Следующий фрагмент отчета позволяет судить о нагрузке мастеров.

Usage

Instantaneous Utilization

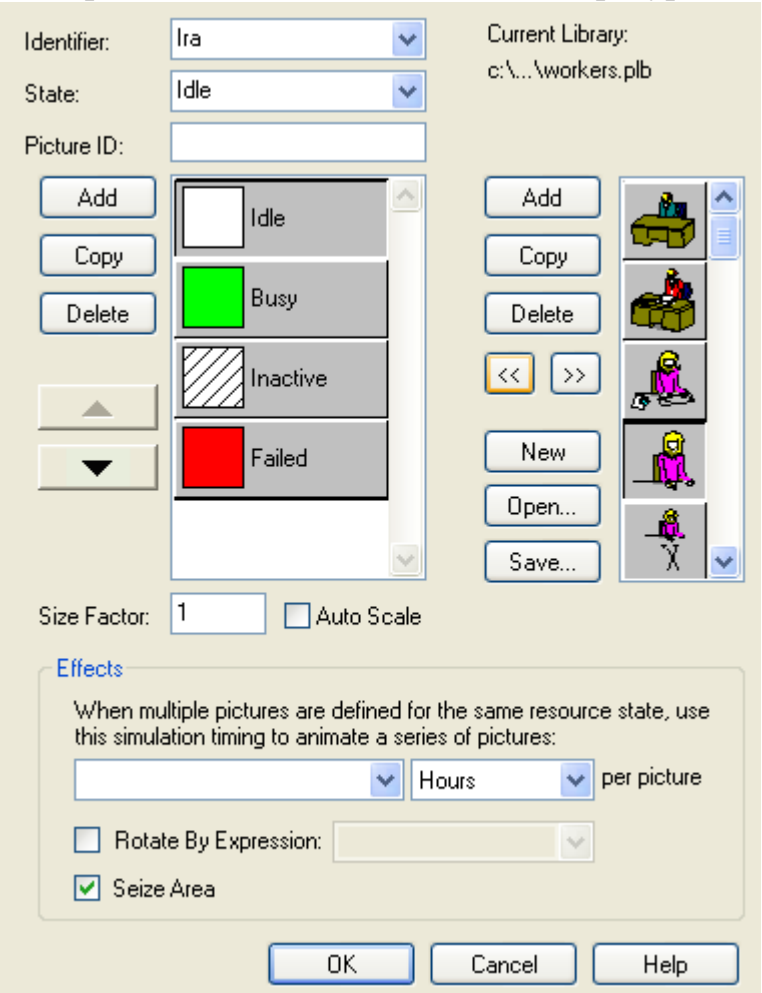
	Average
Ira	0.6553
Lena	0.6174
Masha	0.7942
Nina	0.6176
Rita	0.2519

Из отчета видно, что мастер-универсал Маша имеет наибольшую загрузку среди сотрудников салона.

Экспериментируя с множествами, можно будет повысить эффективность работы мастеров и выручку предприятия. Например, можно обучить одного из женских мастеров мужской стрижке и включить его во множество универсалов. При этом стоит множество мужских мастеров отсортировать так, чтобы мастера-универсалы привлекались в последнюю очередь (в модуле Process указываем правило выбора Preferred Order). Можно проводить и другие эксперименты, отслеживая результаты – выручку, время ожидания в очередях, загрузку мастеров.

Выполним теперь анимацию ресурсов, чтобы наглядно видеть работу парикмахеров.

Выберем элемент панели анимации «ресурс».

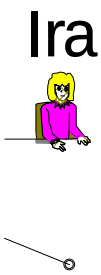


В появившемся окне при необходимости подгрузим графическую библиотеку «Workers». Здесь выберем в правом списке подходящий рисунок. К сожалению, парикмахера среди рисунков нет, поэтому подберем наилучший рисунок и скопируем его кнопкой Copy. Новый рисунок отредактируем.

Теперь выполняем необходимые действия в окне ресурса:

- выбираем идентификатор ресурса Ira;
- размещаем старый рисунок для состояния IDLE (в левом списке выбираем IDLE, в правом - нужный рисунок и нажимаем клавишу <<);
- размещаем новый рисунок с ножницами для состояния BUSY;
- указываем флажок Seize Area (так мы сможем видеть изображение транзакта, которое в настоящий момент обслуживает мастер).

Далее нажимаем Ok и размещаем графический элемент на полотне.



Кружок с пунктирной линией – это и есть Sieze Area, его можно расположить там, где удобно. Надпись с именем парикмахера сделана обычными графическими средствами Arena. Аналогичным образом размещаем графические элементы ресурса для всех остальных мастеров:

При моделировании наглядно видно, кто из мастеров занят и кого из клиентов он обслуживает в настоящий момент.

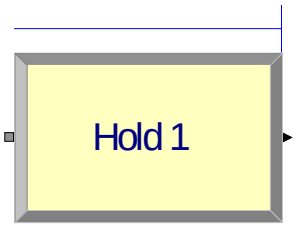
Контрольные вопросы

1) Для чего используют пулы ресурсов?
2) Может ли один ресурс входить в несколько пулов?
3) Что такое правило выбора (Selection Rule)?
4) Если имеется несколько рабочих разной квалификации, как обеспечить направление работы к наиболее квалифицированному свободному рабочему?
5) Как при этом в модуле Process задать время выполнения операции именно выбранным рабочим?
6) Если транзакт захватил ресурс из пула и удерживает его, как сделать так, чтобы в другом модуле транзакт отпустил именно захваченный ресурс?
7) Как сделать загрузку ресурсов пула более или менее равномерной?
8) На предприятии имеется четыре одинаковые единицы ресурса «Станок», как можно визуализировать каждый станок в отдельности?
9) Иван является бригадиром лесорубов и подобно другим руководителям принимает участие в планерках у директора базы. Как отразить тот факт, что Иван при этом остается лесорубом и принимает участие в работе по вырубке леса?
10) С помощью какого модуля создаются пулы ресурсов?
11) Можно ли выбрать для операции несколько единиц одного ресурса из пула?
12) Рабочие объединены в пул и выполняют операции по обработке детали. Однако одну из операций может выполнять только один рабочий, который единственный из всей бригады обладает допуском для работы с опасным оборудованием. Как это можно отразить в модели? Перечислить три возможных способа.

7 Использование шлагбаумов

Модуль **Hold** выполняет роль «шлагбаума» – при определенных обстоятельствах не пропускает транзакты далее, удерживает их. Относится к шаблону Advanced Process.

Обозначение:

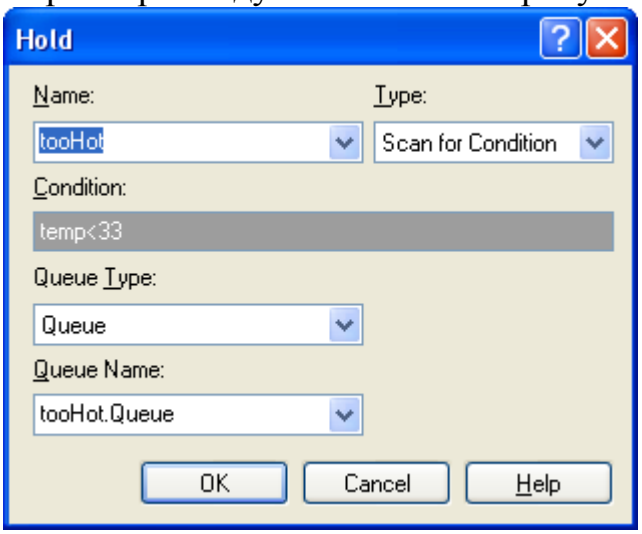


Поскольку модуль задерживает транзакты, то у него имеется очередь, где накапливаются транзакты, которые не могут пройти далее.

Модуль Hold может быть трех типов:

- *ожидать сигнала* (Wait for Signal). Сигнал может быть отправлен из другого места модели с помощью модуля Signal;
- *ожидать условия* (Scan for Condition). Условием может быть логическое выражение. Оно может зависеть от переменных (каждые четыре часа касса делает перерыв, соответствующая переменная устанавливается в единицу и клиенты не могут попасть к кассиру), либо от состояния различных объектов Arena (причал морского порта допускает одновременную швартовку пяти судов, если длина соответствующей очереди равна пяти, то остальные суда вынуждены ждать на рейде);
- *задержка навсегда* Infinite Hold. Шаблон Advanced Process включает модули, позволяющие «вытягивать» транзакты из определенной очереди по заданному условию. Например, можно собирать в течение некоторого времени несколько транзактов-кандидатов в очередь, а затем выбрать из них самого опытного.

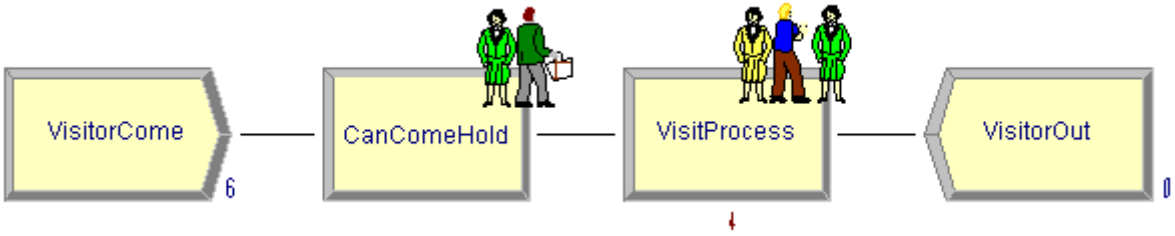
Параметры модуля показаны на рисунке.



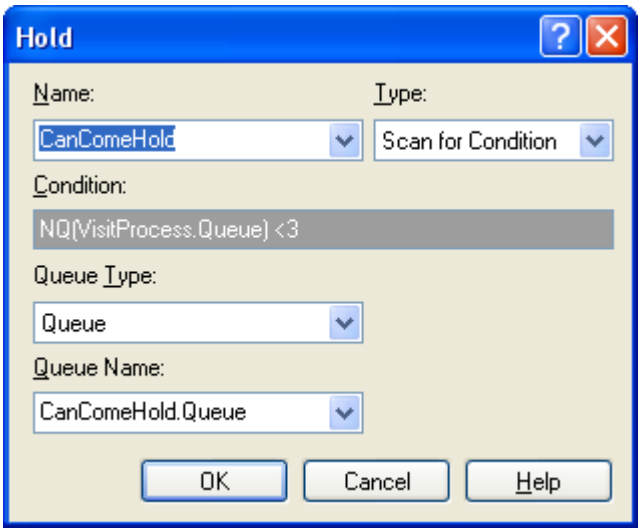
Этот модуль Hold не пропускает транзакты-грузовики, если температура дорожного полотна превышает некоторый предел (33 градуса Цельсия). Значение текущей температуры хранится в переменной *temp*, значение которой может меняться в другом месте модели.

Пример 16. В приемной врача имеется три стула, где располагаются посетители в ожидании приема. Остальные посетители вынуждены ждать в коридоре.

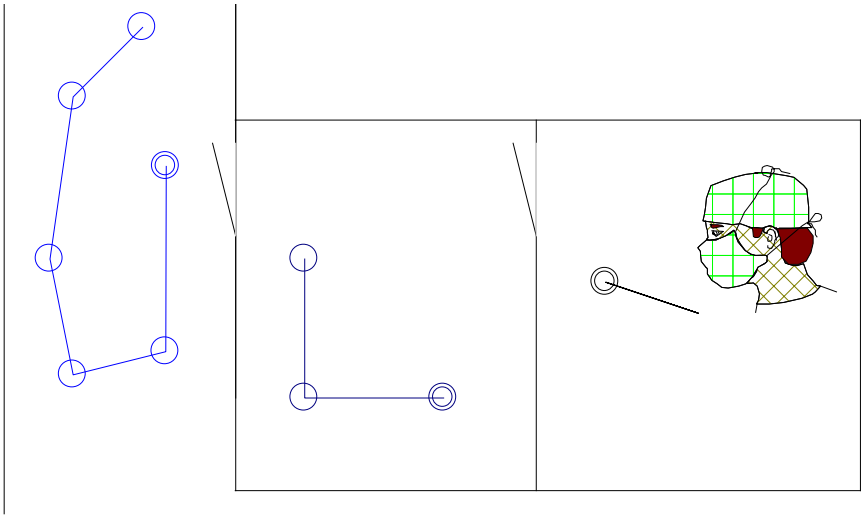
Граф модели показан на рисунке. Модуль Hold проверяет, есть ли свободные места в очереди (длина очереди в кабинете меньше трех). Если есть, транзакт-посетитель переходит в модуль процесса, соответствующий приему и, если врач занят, встает очередь. Очередь возле модуля Hold соответствуют посетителям, ожидающим в коридоре.



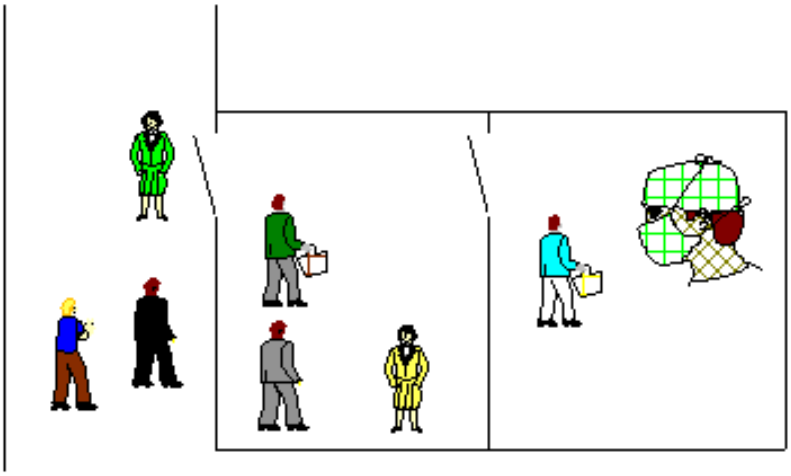
Параметры модуля Hold показаны на рисунке. Тип модуля – «по условию» (Scan for condition). Условие прохода задано логическим выражением *NQ(VisitProcess.Queue)<3*. То есть, число транзактов в очереди должно быть меньше трех.



Добавим анимацию. Для этого нарисуем кабинет, приемную, разместим ресурс (врача) и преобразуем очереди в точечную форму.



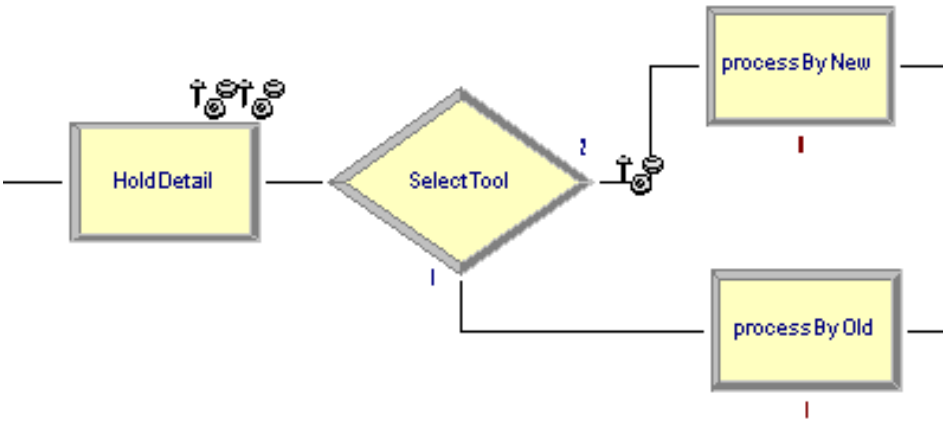
Вид модели во время прогона:



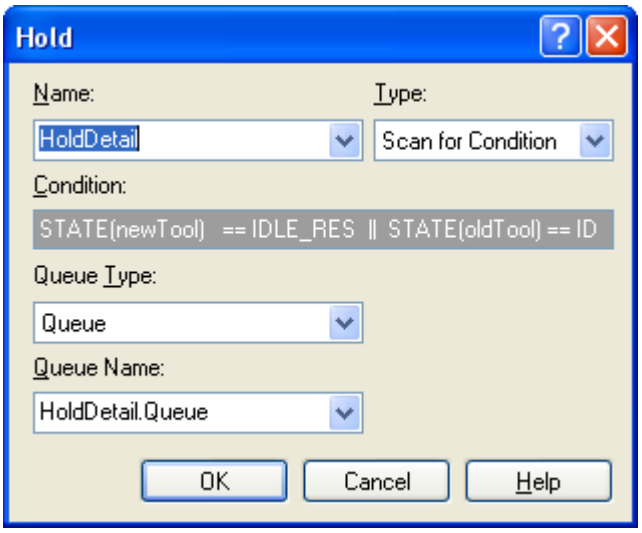
Рассмотрим случай, когда модуль Hold можно использовать для создания общей очереди.

<u>Пример 17.</u> Вернемся к примеру со станками (**Пример 5**). Поступившая деталь будет проходить обработку на более современном станке, но если этот станок занят – деталь будет обработана на менее современном. Ранее, в случае занятости обоих станков мы выводили деталь из модели без обработки. Теперь же реализуем более реалистичный вариант –деталь будет ожидать в очереди, пока не освободится хотя бы один из станков.

Фрагмент модели:



Перед модулем выбора станка установлен модуль Hold, который не пропустит деталь далее, если оба станка заняты.



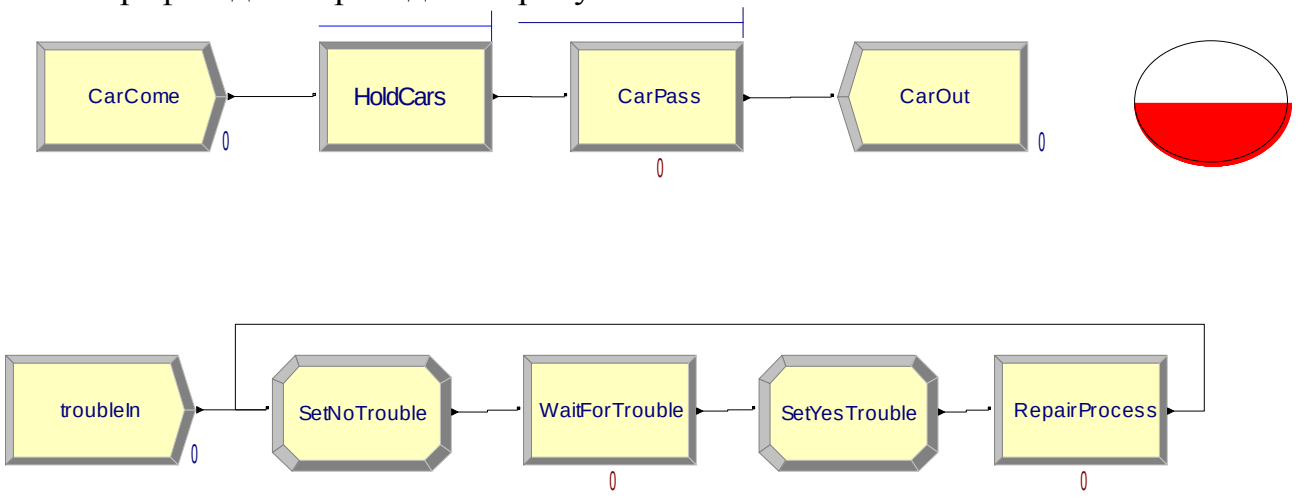
Условие звучит так: «пропустить, если свободен новый станок ИЛИ свободен старый станок».

STATE(newTool) == IDLE_RES || STATE(oldTool) == IDLE_RES

Модуль Hold позволяет строить достаточно гибкие модели, когда поведение компонента зависит от других компонентов системы. Рассмотрим более крупный пример.

Пример 18. На производственном предприятии имеется участок пути, где с помощью автокаров перевозятся материалы. Время от времени, происходит авария, и участок выходит из строя, пока бригада рабочих не ликвидирует последствия аварии. После этого движение автокаров возобновляется.

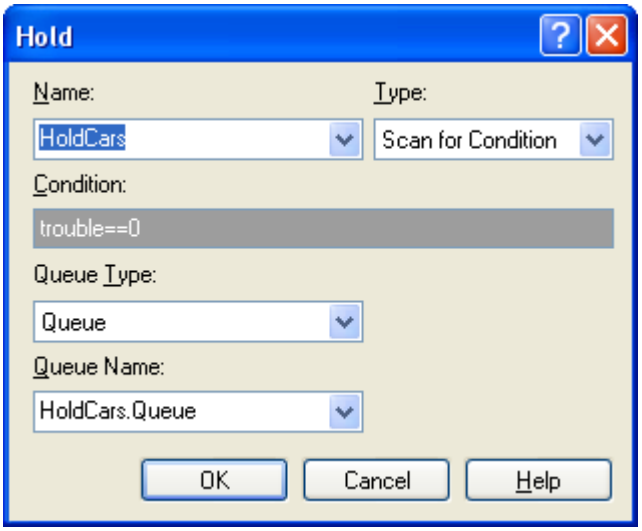
Граф модели приведен на рисунке.



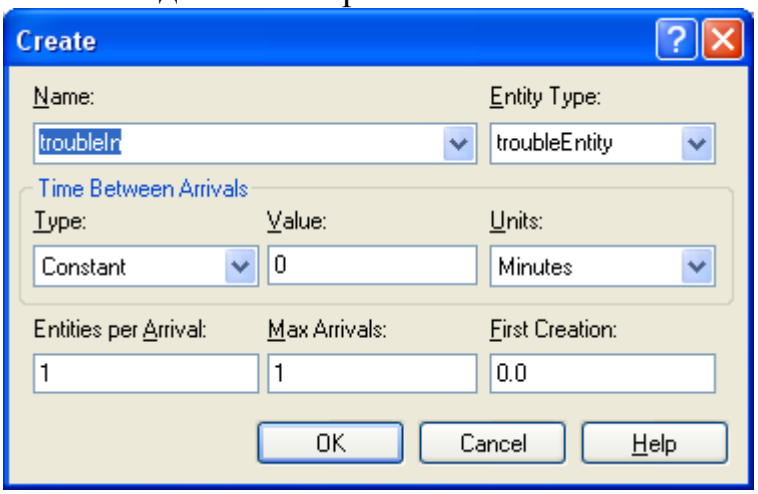
Как видим, имеются два сегмента модели. Верхний отвечает за движение машин. Нижний – моделирует аварию и ее ликвидацию.

Заведем переменную *trouble*, значение которой будет служить признаком наличия аварии в настоящий момент. Если произошла авария и ее пока не ликвидировали, то значение переменной trouble равно 1. Если аварии в настоящий момент нет – значение равно 0.

Модуль Hold будет пропускать машины на участок, если значение переменной *trouble* равно 0:

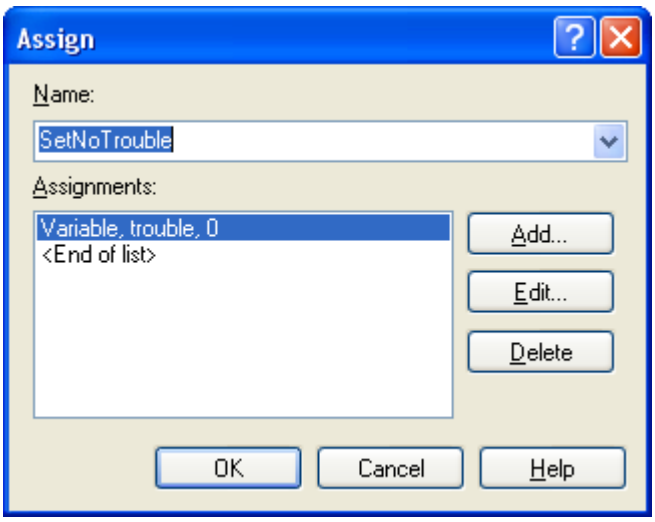


Чтобы моделировать возникновение и ликвидацию аварий введем фиктивный транзакт troubleEntity, который будет циркулировать по нижнему сегменту модели. Транзакт создается модулем Create, в момент времени 0. Поскольку ограничитель Max Arrivals установлен равным единице, то будет создан только один такой транзакт.

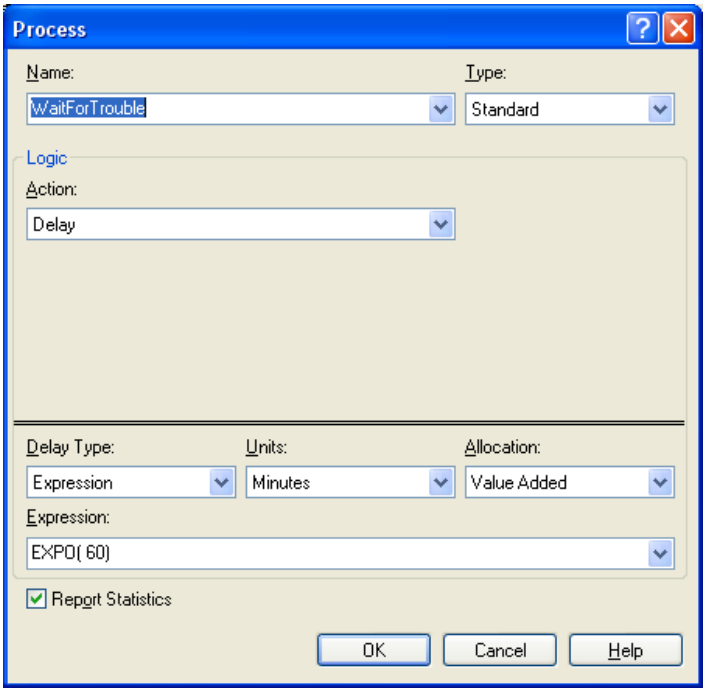


Для анимации добавим светофор. Используем элемент Level, где в поле Expression укажем имя переменной trouble и выберем форму круга для этой фигуры.

Проходя следующий модуль (Assign) транзакт устанавливает значение переменной *trouble* в ноль. То есть система приводится в состояние «аварии нет».



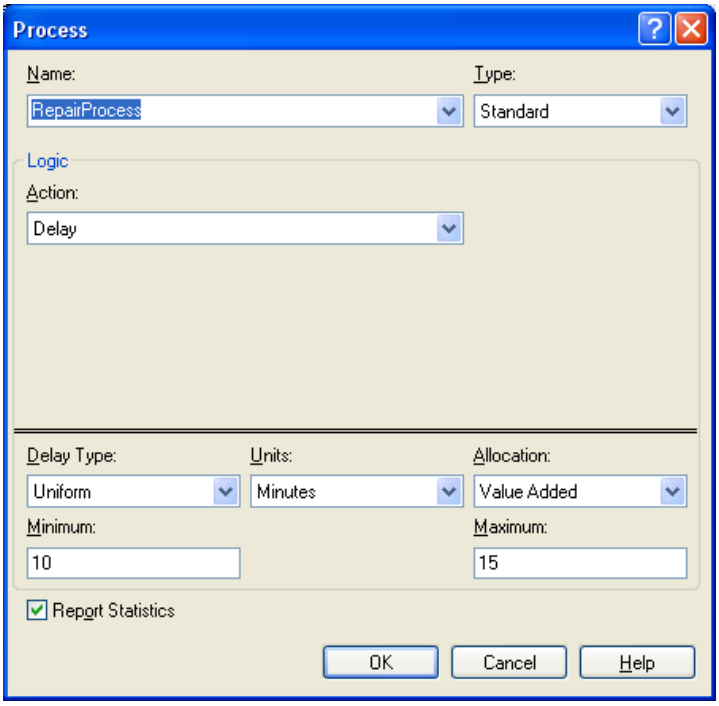
Далее транзакт следует в модуль Process с называнием WaitForTrouble. Смысл «обслуживания» транзакта состоит в том, что участок эксплуатируется до появление аварии. Окончание обслуживания означает, что произошла авария.



Далее транзакт следует в модуль Assign, где, установив значение переменной *trouble* в единицу, объявляет системе о наличии аварии. Теперь модуль Hold не пропускает транзакты-машины.

Далее транзакт troubleEntity попадает в модуль Process, символизирующий ремонт участка силами бригады рабочих. Модуль имеет тип Delay,

бригада не рассматривается как ресурс, поскольку двух одновременных аварий возникнуть не может и конкуренция за бригаду рабочих невозможна.



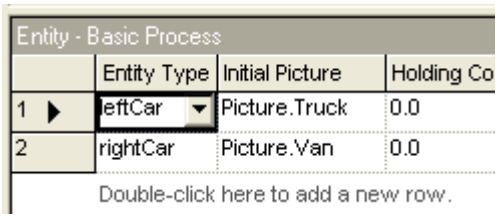
Задача 2. Автомобили прибывают к железнодорожному переезду примерно раз в минуту (закон распределения экспоненциальный). Чтобы пересечь переезд автомобиль тратит 10-20 секунд (закон распределения равномерный). Раз в 10 минут (закон распределения экспоненциальный) приезжает поезд, которому требуется 3 минуты (экспоненциальный), чтобы миновать переезд.

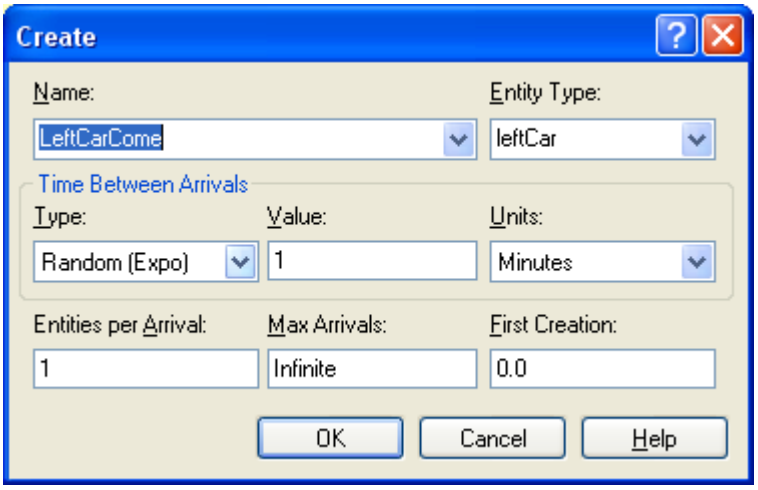
Провести моделирование системы, используя модуль Hold и переменную, означающую наличие поезда.

Пример 19. Имеется мост, движение автомобилей по которому возможно только в одну сторону в одно время. При этом на мосту помещаются 5 автомобилей. Чтобы пересечь мост нужно ровно две минуты. Автомобили прибывают к каждому концу моста раз в минуту (закон распределения экспоненциальный).

Для решения задачи необходимо определиться с транзактами. В модели будут два типа транзактов:

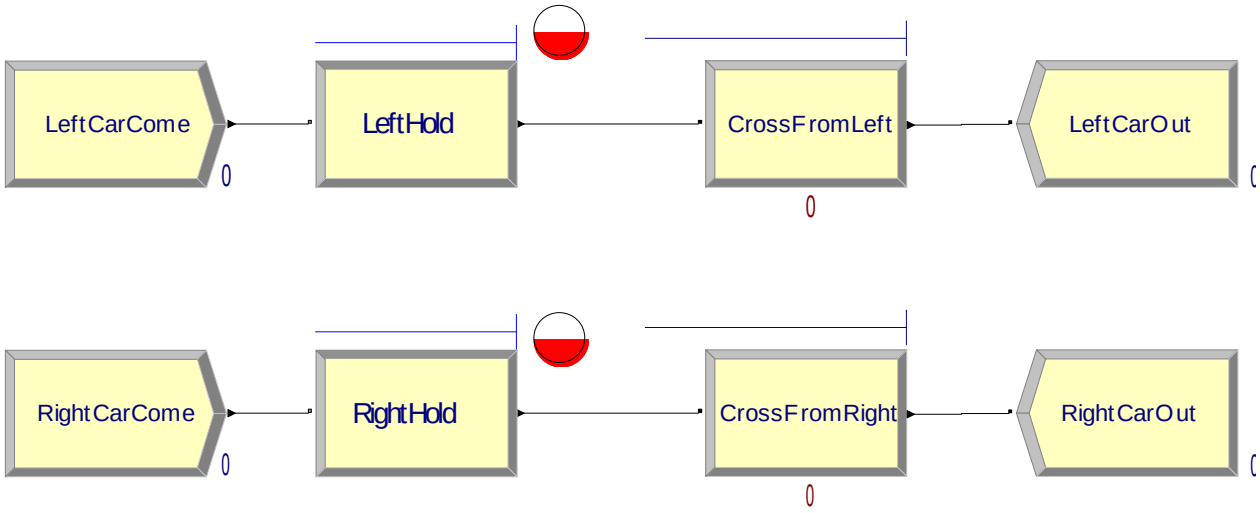
- автомобили, прибывающие к левому концу моста leftCar;
- автомобили, прибывающие к правому концу моста rightCar.





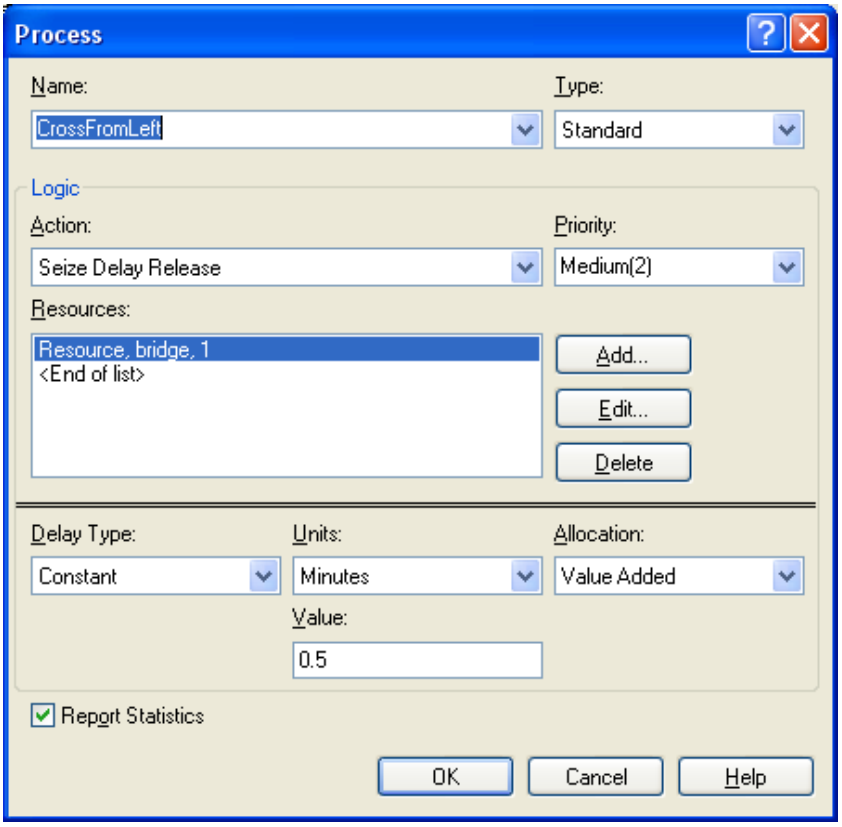
В качестве ресурса установим «место на мосту», причем зададим мощность, равную 5. Мы можем не задумываться о том, что автомобили пересекают мост один за другим, поскольку время пересечения фиксировано и обгонять друг-друга автомобили не могут. Важно лишь общее число автомобилей, которые могут оказаться на мосту.

Имеются два сегмента модели, верхний демонстрирует проезд автомобилей слева направо, нижний – справа налево.

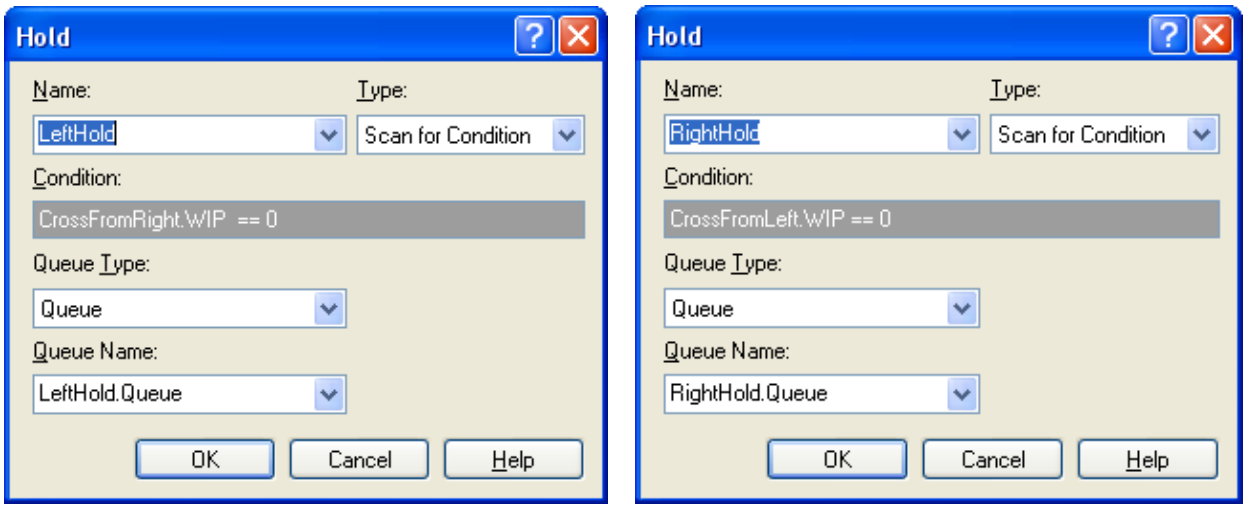


Пересечение моста – это процесс.

Поскольку мост может пересекать несколько автомобилей одновременно, то невозможно моделировать его как конкуренцию за ресурс – то есть с помощью процесса. Если автомобиль подъезжает к мосту слева, а в этот момент мост пересекает автомобиль справа, левый автомобиль не должен вступать на мост, даже если на мосту есть свободное место.



Таким образом, нужна пара модулей Hold. Первый из них (LeftHold) не допустит въезда автомобилей на мост слева, если кто-то пересекает его справа. Аналогично, второй модуль RightHold не допустит автомобиль справа, если другие автомобили пересекают мост слева.



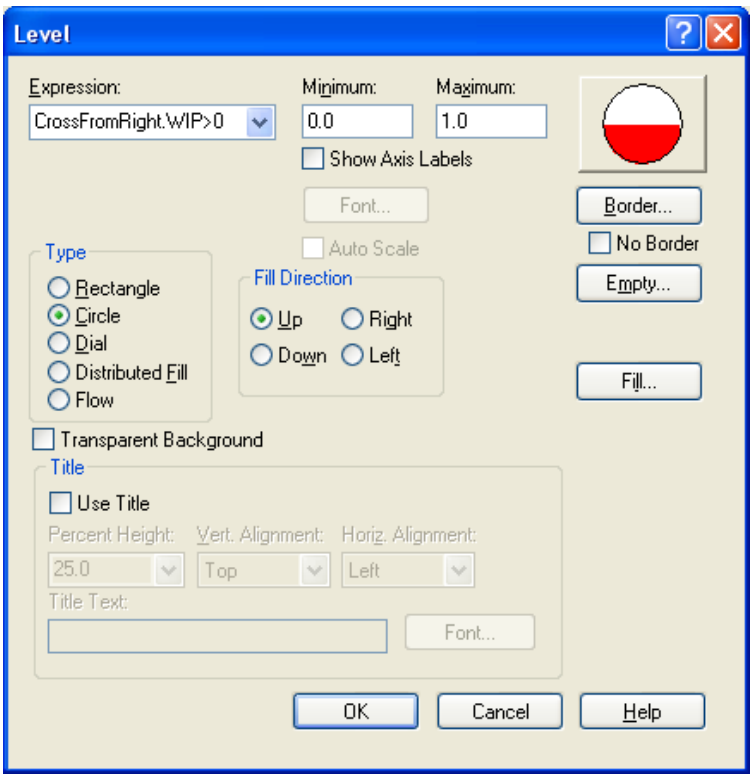
Модули Hold имеют тип Scan for Condition (по условию). Модуль LeftHold, призванный ограничивать автомобили слева имеет логическое выражение для условия пропуска:

CrossFromRight.WIP ==0

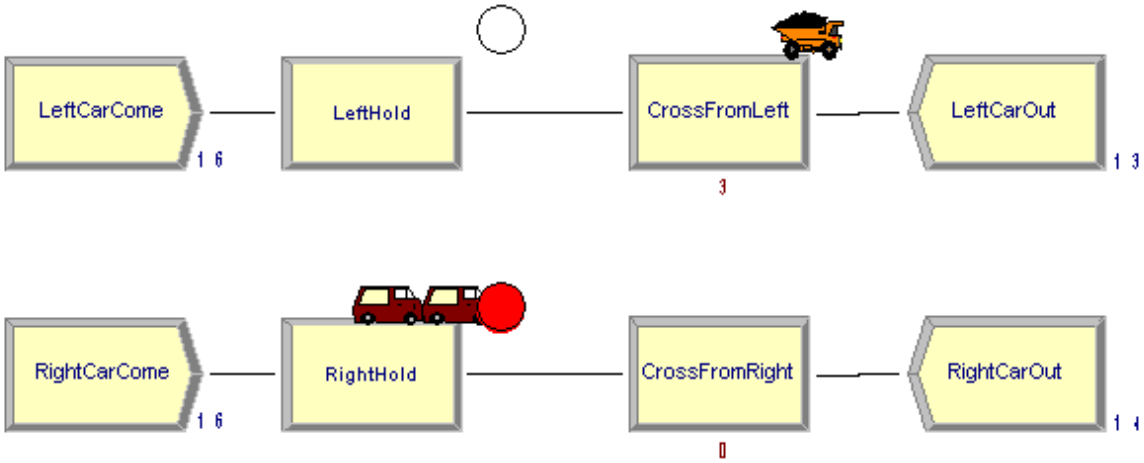
То есть модуль Process, отвечающий за пересечение моста справа не должен обрабатывать в настоящий момент ни одного транзакта. Только в этом случае левый автомобиль может начинать пересечение моста.

Модуль RightHold работает аналогично, но проверяется число транзактов в модуле CrossFromLeft.

Для повышения наглядности изобразим светофоры в виде элементов Level:



Вид модели во время прогона:



Задача 3. Решить предыдущий пример (**Пример 19**) в условиях использования светофора, который делит время между правым или левым потоками. Разумеется, автомобиль не должен въезжать на мост, если с противоположной стороны автомобили заканчивают движение, даже при зеленом свете. После того, как задача со светофором будет реализована, следует сравнить эффективность двух систем – без регулирования и с регулированием – с точки зрения времени ожидания автомобилей перед мостом.

Задача 4.Отдел фирмы, обрабатывает документы, означающие приход и уход денежных средств. Документы поступают в разные моменты времени. В каждом документе прописана определенная сумма (формируется случайным образом), которая либо поступает на счет фирмы, либо должна быть уплачена. После поступления денежных средств (обработка приходного документа) баланс на счете увеличивается. Если баланс не позволяет оплатить расходный документ – расходный документ ожидает. Возможно, следует сначала оплачивать более мелкие суммы. Вероятно, стоит учесть время обработки документа сотрудником.

Необходимо отследить, насколько часто задерживается оплата.

Указание – следует использовать модули Assign, атрибуты транзактов, глобальную переменную для баланса, модуль Hold.

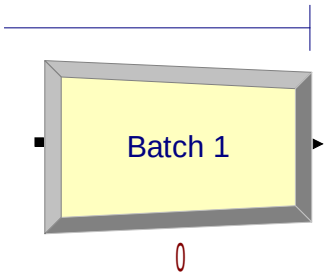
Контрольные вопросы

1) Какие бывают типы модуля Hold?
2) Что делает модуль Hold?
3) Каков смысл использования задержки транзакта навсегда?
4) Почему у модуля Hold имеется очередь?
5) Как работает модуль Hold с вариантом «по условию» (by condition)?
6) Как организовать задержку всех транзактов-автомобилей, если железнодорожный переезд сейчас пересекает поезд?
7) Какие параметры транзакта и модулей могут использоваться в условии модуля Hold?
8) В каком случае модуль Hold в варианте «по условию» пропускает транзакты?
9) Где в системе Arena можно найти модуль Hold?

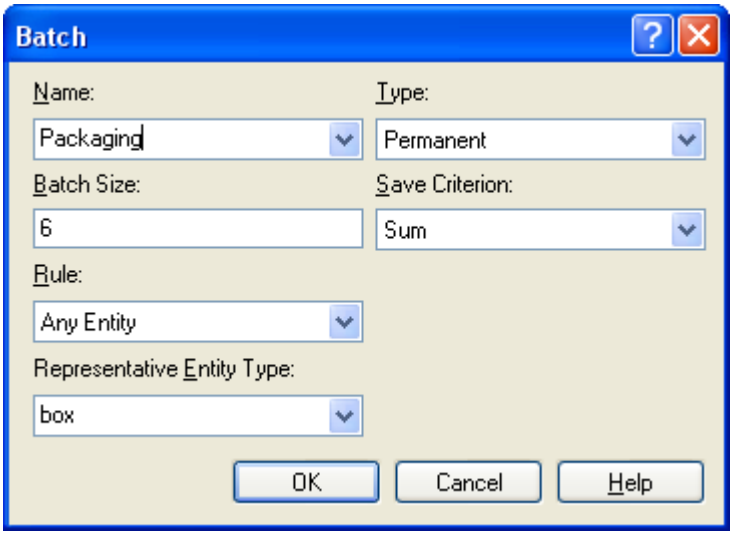
8 Слияние, расщепление и синхронизация транзактов

Модуль соединения **Batch** – производит объединение (временное или постоянное) заданного числа транзактов в один транзакт. Модуль имеет очередь, где поступающие траназкты ожидают, до тех пор пока не будет набрано необходимое число транзактов. После этого транзакт-сборка последует по модели далее.

Обозначение:



Параметры модуля Batch показаны на рисунке.



Важнейшими свойствами являются:

- *тип* (Type). Постоянное соединение (Permanent) не предполагает дальнейшего разделения транзакта-сборки на части. Пример постоянного соединения – сборка агрегата из деталей. Далее по модели следует агрегат. Временное соединение (Temporary) впоследствии может быть разобрано. Пример – пассажиры заполняют вагон, вагон следует до станции, пассажиры покидают вагон (для разборки временного соединения используют модуль Separate);

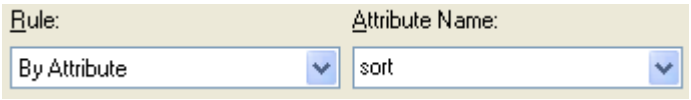
- *размер* (Batch Size). Это число объединяемых транзактов;

- *результирующий тип транзакта* (Representative Entity Type). Это тип транзакта-сборки. Например, детали собираются вместе и результирующим транзактом является собранный агрегат (это будет видно наглядно, поскольку этому типу транзактов можно задать свою картинку). Еще пример – бу-

тылки вина комплектуются по 6 штук и помещаются в ящик, который и является транзактом-сборкой.

- *правило объединения* (Rule). Если выбрано правило Any Entity модуль будет объединять любые транзакты, даже транзакты разного типа. Если же указать правило «в соответствии со значением атрибута» (by attribute) и задать имя атрибута, то объединяться будут только транзакты, с одинаковым значением этого атрибута. Например, можно выбрать атрибут Entity.Type (тип транзакта) и тогда комплектоваться в группы будут транзакты-детали одного типа.

Еще пример – на заводе с традиционной технологией изготовления вина готовые бутылки поступают в цех упаковки. Не следует собирать в один ящик бутылки разных сортов. Нужно задать правило «по атрибуту» и указать атрибут, соответствующий сорту вина. Теперь будут формироваться отдельно ящики шампанского брют, полусухого шампанского и т.д.



Транзакты-бутылки будут ожидать в очереди пока не соберется нужное число бутылок какого-то сорта.

Еще один распространенный случай использования правила – указывать в качестве атрибута серийный номер транзакта Entity.SerialNumber. Поскольку при копировании транзактов [20] это номер сохраняется, то появится возможность собирать вместе копии одного транзакта.

- *правило сохранение значений (Save Criterion)* Позволяет, например, суммировать затраты на обработку всех собираемых транзактов-деталей, чтобы рассчитать себестоимость полученного агрегата.

Модуль разделения (дублирования) **Separate**. Снимает с транзакта заданное число копий, либо разделяет ранее соединенную модулем Batch транзакт-сборку.

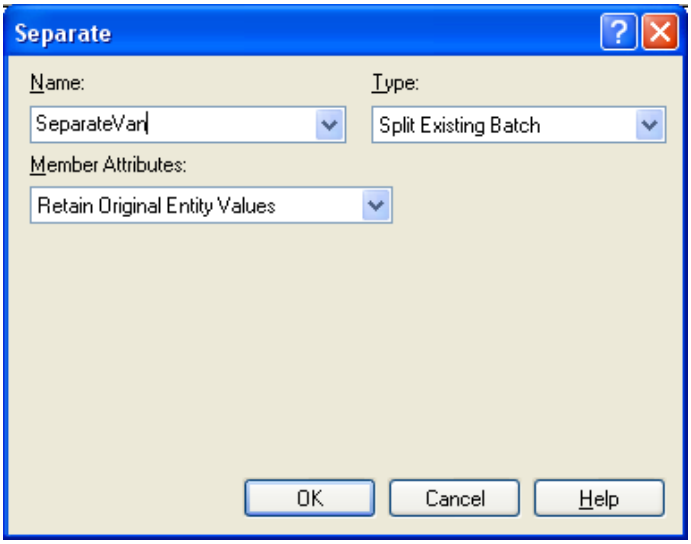
Обозначение:

Параметры модуля показаны на рисунке. Здесь выбран тип «Split Existing Batch» (расщепить ранее собранный транзакт-сборку).

Транзакт-сборка (полученный в модуле Batch с типом временного соединения, Temporary), попав модуль Separate, будет разобран. Пример – пассажиры автобуса покинут его после прибытия на место назначения.

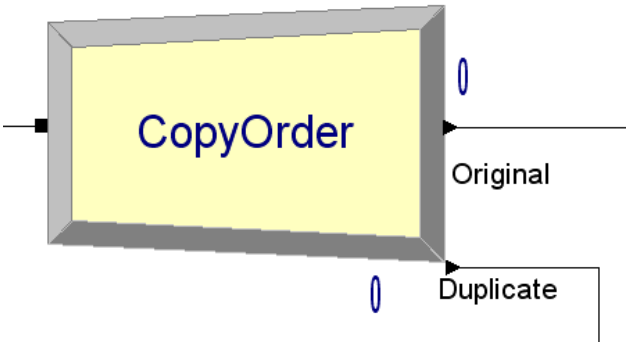
Модуль позволяет по разному разносить значения атрибутов по разделяемым транзактам.

[20] Копирование осуществляется модулем Separate



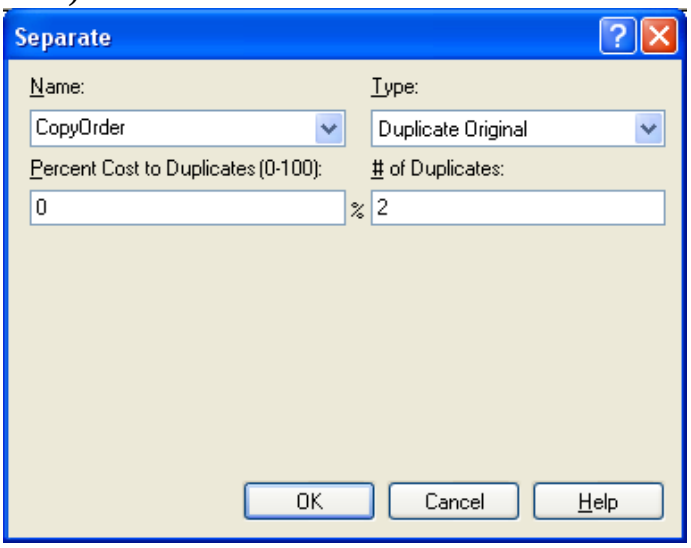
Другой тип модуля Separate – дублирование транзакта (Duplicate Original). При попадании транзакта модуль создает заданное число его копий. Копии сохраняют серийный номер транзакта-родителя и наследуют все его атрибуты. Накопленная на родительском транзакте стоимость могут быть перенесены (полностью или частично) на новые транзакты.

Обозначение:



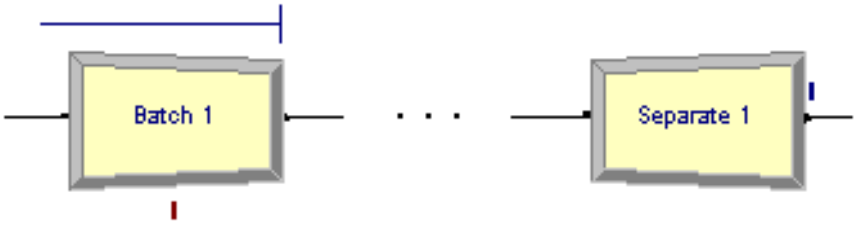
Модуль имеет два выхода, один – для исходного транзакта (Original), второй – для транзактов-копий (Duplicate).

Среди параметров модуля основной – число создаваемых копий (# of Duplicates).

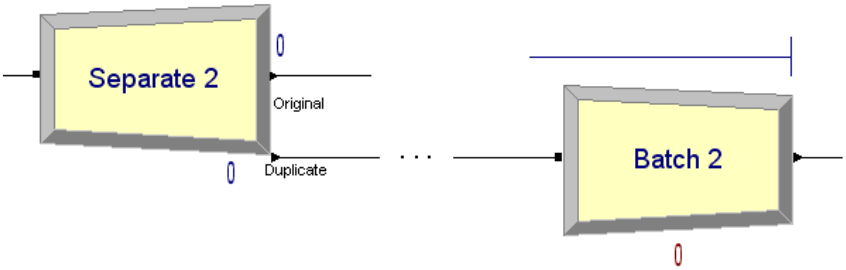


Часто модули Batch и Separate используются совместно.

Первый вариант: транзакты объединяются во временную сборку (тип Temporary) в модуле Batch, а затем после прохождение какой-то обработки разъединяются в модуле Separate (тип Split Existing Batch). Например, пассажиры занимают места в автобусе, автобус совершает поездку, пассажиры выходят из автобуса:

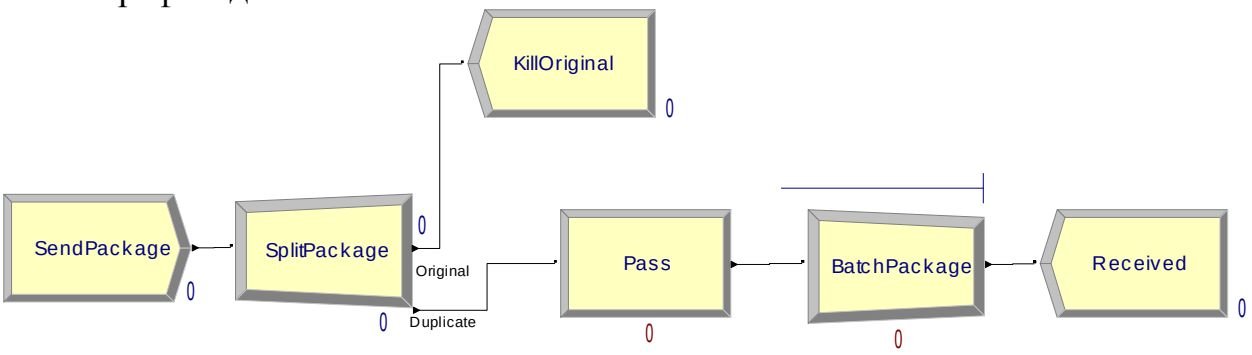


Второй вариант: модуль Separate производит копирование (тип Duplicate Original), затем копии вновь собираются в один в модуле Batch, постоянное соединение (тип Permanent):

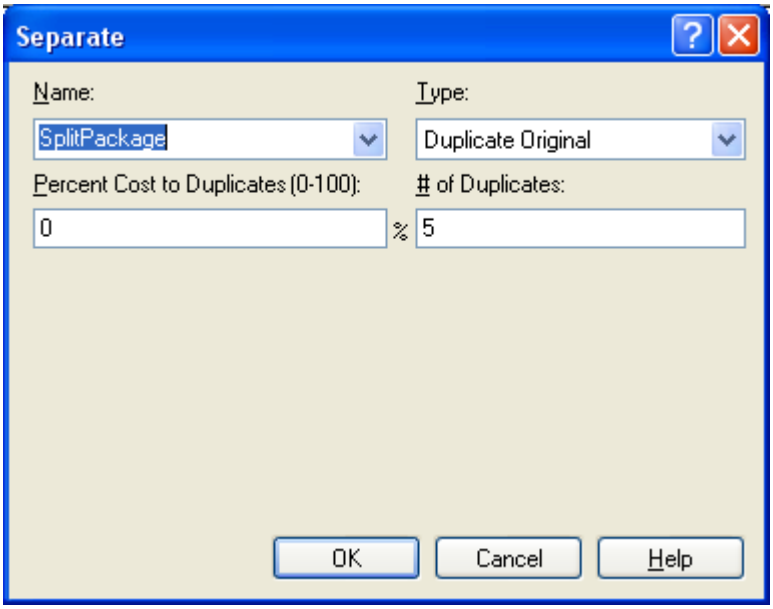


Пример 20. При передаче данных через сегмент компьютерной сети транзакты-пакеты разделяются, поскольку протокол нижнего уровня ограничивает размер пакета, пропускаемого через этот сегмент. На станции назначения пакеты вновь собираются. Разумеется, должны собираться исходные части каждого конкретного пакета, поэтому при сборке в модуле Batch используется атрибут SerialNumber.

Граф модели:

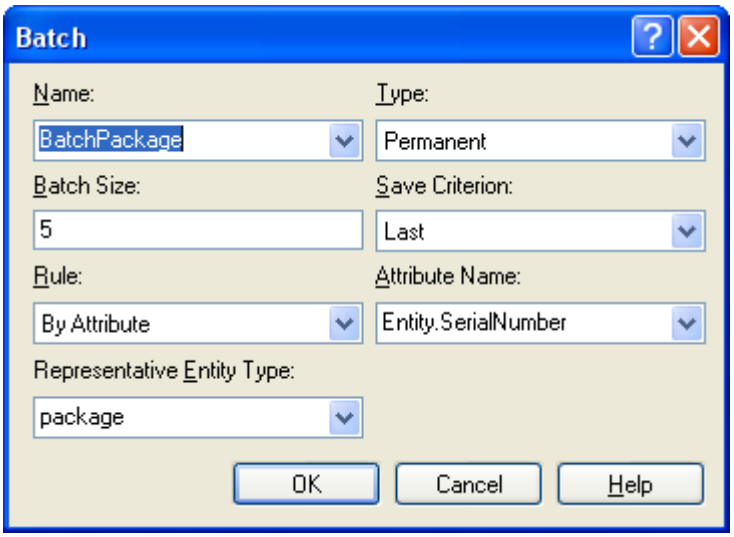


В модуле Separate происходит фрагментация сетевого пакета, его разделение на 5 частей (исходя из разницы в размерах пакетов для сетевых протоколов). Исходный транзакт передается в модуль Dispose и выводится из модели. Транзакты-копии следуют далее.



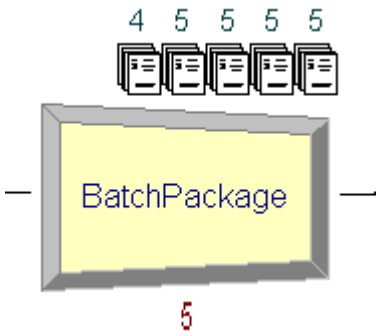
Транзакты-фрагменты по отдельности передаются через сеть. Поскольку процессы передачи пакетов по сетям носят случайный характер, то на конечную станцию будут поступать множество фрагментов из разных исходных пакетов, причем фрагменты каждого пакета могут приходить в произвольном порядке.

Модуль Batch производит объединение пакетов по 5 штук, причем объединяются именно фрагменты одного пакета, а не разных. Это достигается благодаря выбранному правилу By Attribute. В качестве атрибута для сборки используется серийный номер транзакта Entity.SerialNumber.



Ранее при дублировании транзакта (при фрагментации в модуле Separate) все копии получили тот же серийный номер, что и у родителя.

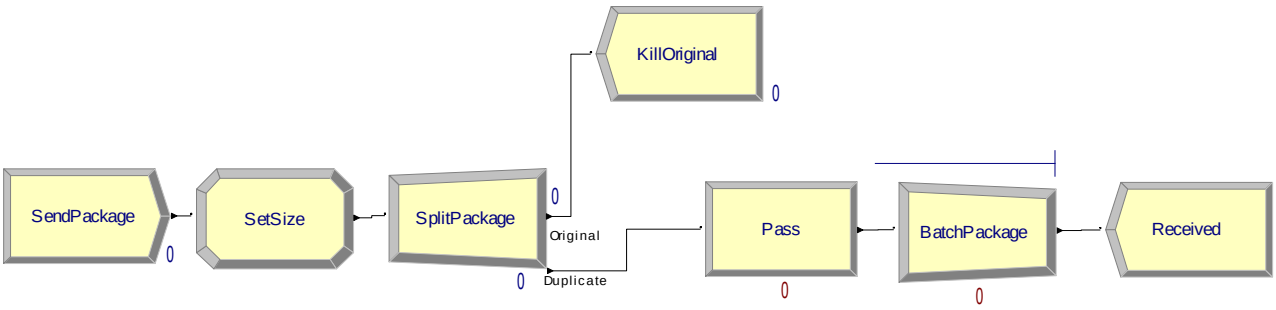
На рисунке показан модуль Batch, осуществляющий сборку фрагментов. Возле изображений транзактов нанесены их серийные номера. Мы видим, что в очереди имеются фрагменты двух разных пакетов.



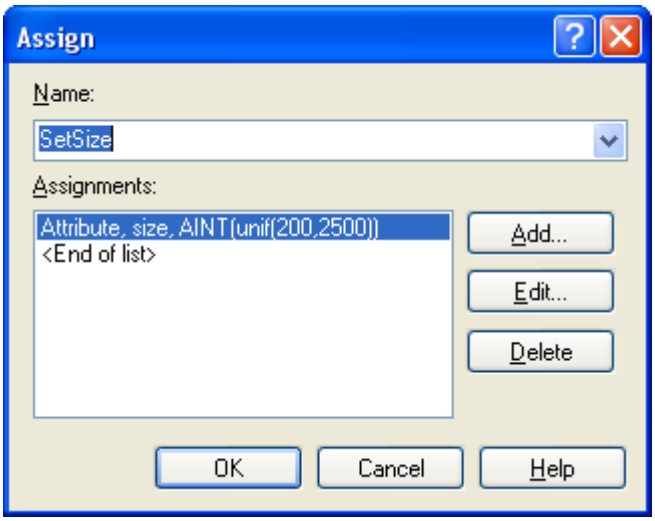
Описанная модель в упрощенном виде иллюстрирует некоторые принципы работы таких протоколов, как IP.

Пример 21. Сделаем прошлый пример (**Пример 20**) немного более реалистичным. Будем учитывать, что пакеты, передаваемые по сети, имеют разный размер, следовательно и число фрагментов может быть различным.

Заведем специальный атрибут *size* для размера исходного передаваемого пакета в байтах (для установки атрибута добавим в модель модуль Assign). Новая структура модели показана на рисунке:



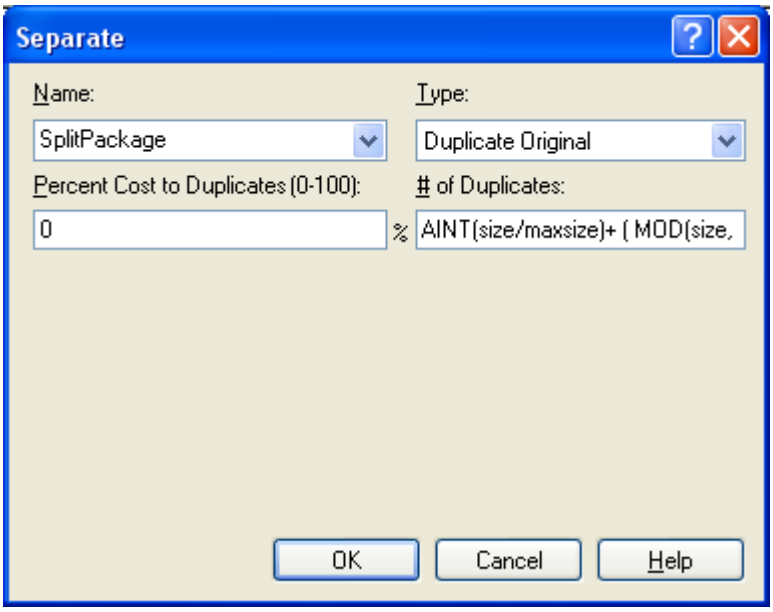
Параметры модуля Assign, где устанавливается случайным образом длина пакета в байтах показаны на рисунке:



В переменной *MAXSIZE* будем хранить максимально возможный размер фрагмента для передачи по сети. Соответственно теперь, пакет разбива-

ется на необходимое число фрагментов. При *MAXSIZE=500* пакет размером в 650 байт будет разбит на два фрагмента. Пакет размером 1200 байт будет разбит на 3 фрагмента и т.д.

Расчет необходимого числа фрагментов производится в модуле Separate:

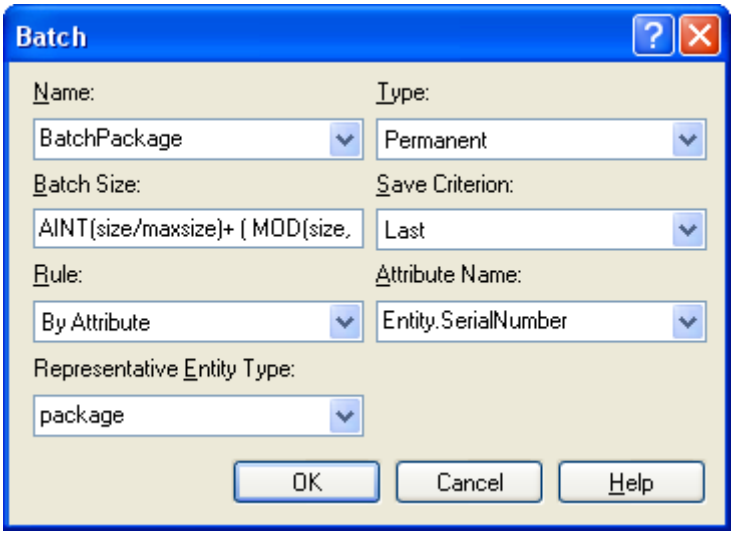


Число пакетов равно:

AINT(size/maxsize)+ (MOD(size,maxsize) >0)

Первое слагаемое округляет в меньшую сторону результат деления, а второе слагаемое принимает значение 1, если у деления имеется остаток и 0, если остатка нет. В синтаксисе Arena у логических операций результат 0 соответствует значению ЛОЖЬ, результат 1 соответствует значению ПРАВДА.

Аналогичный расчет производится и в модуле Batch.

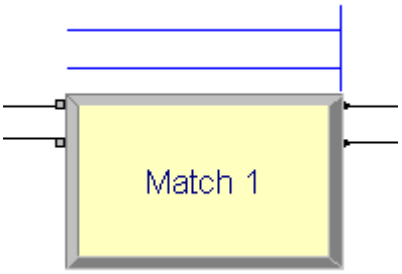


Таким образом, разделение и объединение транзактов может опираться на значения их атрибутов.

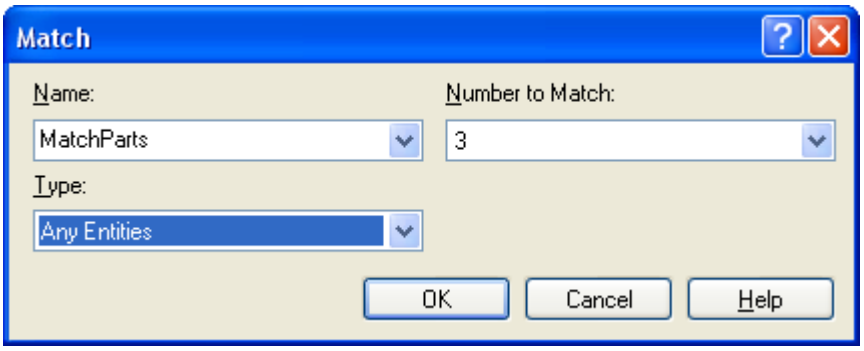
Модуль **Match** осуществляет синхронизацию двух или более (до пяти) транзактов. Имеет несколько входов и соответствующее число выходов. Поступивший на вход транзакт ожидает, пока на другой вход (или на все другие входы, если их несколько), поступят транзакты. Как только набор транзактов на всех входах собран – все они одновременно продолжают движение, причем каждый на свой выход.

Модуль имеет две или более очереди (по числу входов). В этих очередях поступающие транзакты ждут синхронизации.

Обозначение:



Пример использования – имеется три различные части одного агрегата, которые вместе должны пройти какую-то операцию. Эти три части должнs поступить на эту операцию одновременно, что достигается с помощью модуля синхронизации Match:



Параметры модуля:

- *число синхронизируемых транзактов* (Number to Match). Это число входов, выходов, очередей. В приведенном выше примере должны поступить три различные части какого-то агрегата (например: кузов, двигатель, рама);

- *тип объединения (Type).*

Тип может принимать одно из двух значений:

«Any Entities» - любые поступившие транзакты будут производиться синхронизации. Так, в предыдущем примере не важно, какой именно кузов поступил, детали являются стандартными.

«By Attribute» - будут синхронизироваться транзакты, имеющие одинаковое значения заданного атрибута. Например, должны пройти дальше детали, которые относятся к одному конкретному заказу (синхронизация по атрибуту «Номер заказа»).

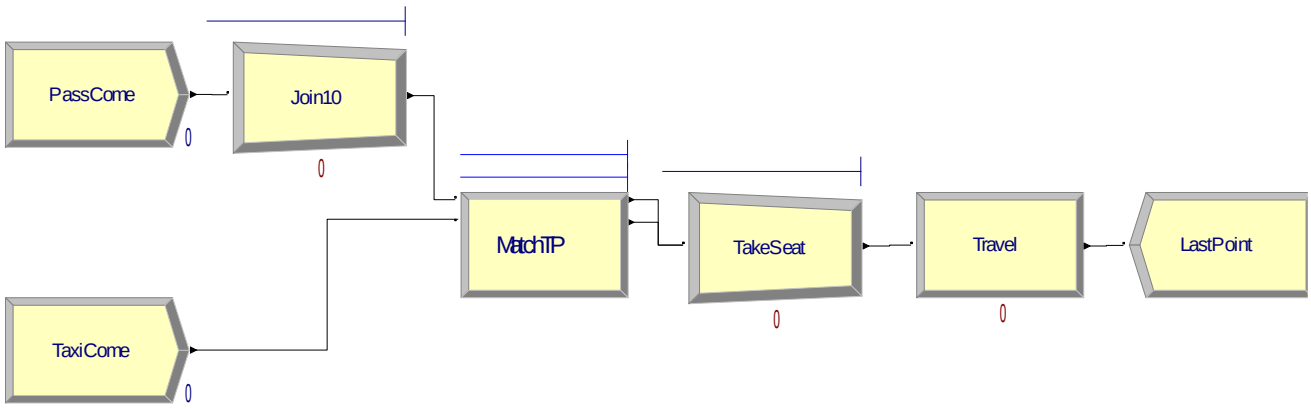
Пример 22. Требуется построить модель, имитирующую работу маршрутных такси, изучить работу системы (определить среднее время ожидания пассажиром такси, среднее время пребывания такси на маршруте, среднее время стоянки такси, число обслуживаемых пассажиров) [21]. Параметры модели представлены в таблице.

Параметр	Закон распределения	Параметры закона
Интервал прихода пассажиров	экспоненциальный	Мат. ожидание = 1 мин
Интервал прихода маршрутного такси	экспоненциальный	Мат ожидание = 10 минут
Время поездки до конечной остановки	экспоненциальный	Мат ожидание =20 мин;
Число пассажиров в такси		10

Иными словами, на остановку раз в 10 минут подъезжают маршрутные такси. Очередное такси подается к месту посадки, когда предыдущее отправляется на маршрут. Также к остановке подходят пассажиры, раз в минуту. Пассажиры занимают место в такси, которое отправляется на маршрут только когда будет заполнено полностью.

Очевидно, что имеются два типа транзактов: такси и пассажиры.

Граф модели показан на рисунке.

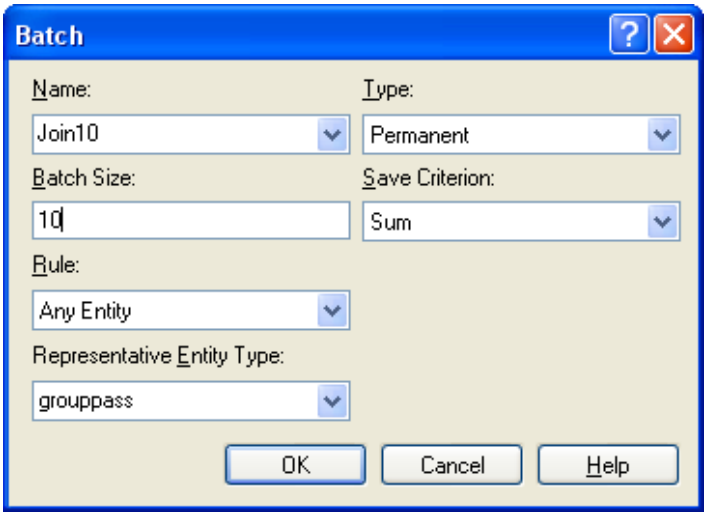


Транзакты обоих типов создаются соответствующими модулями Create. При этом транзакты-такси следуют в модуль Match, а пассажиры должны сначала собраться в группу из десяти человек. Для этой цели установлен модуль Batch с именем Join10.

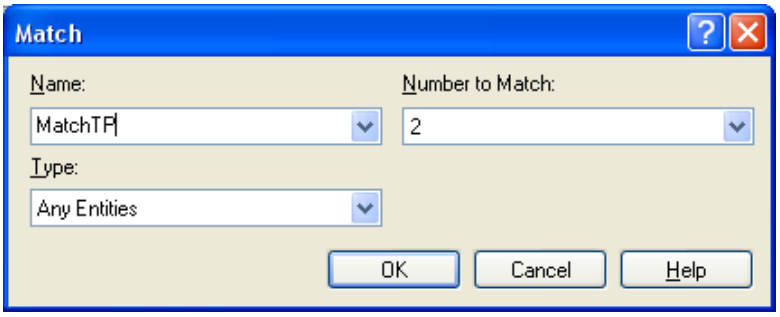
Параметры модуля Batch показаны на рисунке. Объединение является постоянным (Permanent). Должны подойти 10 пассажиров (Batch size 10). Исходящим транзактом является группа пассажиров, готовая к посадке в такси и поездке.

Модуль имеет очередь – там собираются пассажиры, пока их не станет 10 и не будет сформирована группа.

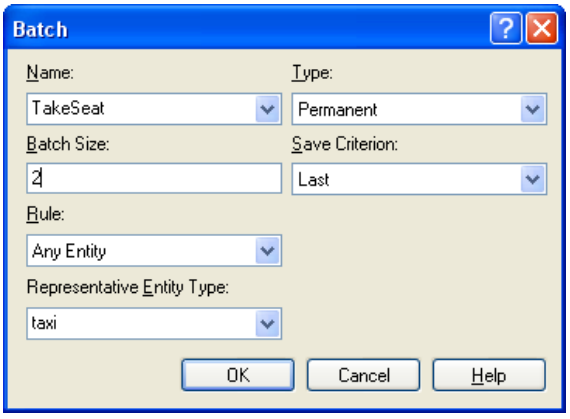
[21] Задача взята из книги Емельянов А.А. Имитационное моделирование в управлении рисками – СПб.: СПбГИЭА, 2000. – 376 с.



Ключевым для этой модели является модуль Match, обеспечивающий условие для отъезда такси – наличие машины и десяти пассажиров. Одновременно может быть заполнена только одна из его очередей – или пассажиры ждут такси, или такси ожидает, пока появятся 10 пассажиров. Параметры модуля показаны на рисунке. Число синхронизируемых транзактов равно двум – это такси и группа пассажиров.



Второй модуль Batch с названием TakeSeat символизирует посадку пассажиров в такси. Исходящим транзактом является такси. Размер равен двум, это значит должны объединиться такси и группа пассажиров. Благодаря тому, что модуль Match подаст два этих транзакта одновременно, будет гарантирована правильность объединения.



После объединения транзактов (пассажиры сели в такси) машина отправляется на маршрут (модуль Process).

Контрольные вопросы

1) Могут ли одновременно все очереди модуля Match содержать транзакты, если установлено правило синхронизации «Any Entity»?
2) К ЗАГСу на автомобилях подъезжают женихи и невесты. Далее осуществляется синхронизация перед началом церемонии бракосочетания. Какое правило синхронизации следует использовать?
3) Сколько выходов может иметь модуль Separate?
4) В чем особенность атрибута Serial Number?
5) Какие два наиболее частых случая совместного использования модулей Separate и Batch?
6) Сколько очередей имеет модуль Match?
7) Зачем нужна очередь в модуле Batch?
8) Сколько очередей имеет модуль Separate?
9) Объединяет ли модуль Match транзакты?
10) Что означает правило объединения (Rule) в модуле Batch?
11) Как сделать так, чтобы результатом объединения нескольких транзактов-узлов был транзакт-автомобиль?
12) Каким модулем можно разделить временно объединенные в группу транзакты?

9 Методы передачи транзактов. Использование станций

Шаблон Advanced Transfer предлагает разработчикам модели несколько более сложных способов передачи транзактов для отражения моделирования сложных производственных и управленческих процессов.

В частности, предлагаются такие конструкции, как станции, транспортеры, конвейеры и соответствующий набор модулей [22].

Станция – это физическое или логическое место, где может находиться транзакт. Например, это может быть цех или склад. Соответственно можно построить путь транзакта через несколько станций.

Преимущество использования станций заключается в возможности разбить модель на несколько более мелких частей, каждая из которых работает относительно независимо.

Например, имеется несколько типов заготовок, каждая из которых, прежде чем стать готовым изделием, проходит свою собственную последовательность различных цехов, оказываясь, в конечном итоге, на складе готовой продукции. В рамках каждого фрагмента модели можно моделировать, например, токарный цех, не задумываясь о дальнейшем пути конкретной заготовки.

Перемещение между станциями может занимать определенное время, что отражает реальную ситуацию.

Первым модулем, который следует рассмотреть, является **Station**, поступление на станцию [23].

Обозначение:

Когда транзакт попадает в этот модуль, он оказывается на определенной станции.

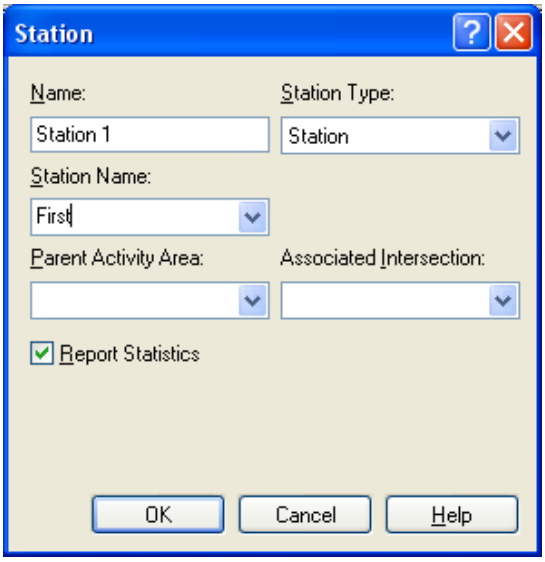
Например, требующие ремонта телевизоры оказываются в пункте приема, куда их приносят клиенты. Далее, если по документам телевизор все еще подлежит гарантии, он может быть направлен на экспертизу.

Окно параметров модуля Station изображено ниже. Основным полем здесь является Station – то есть имя станции, на которую попадает транзакт, проходя модуль. В данном случае, транзакт поступит на станцию «First».

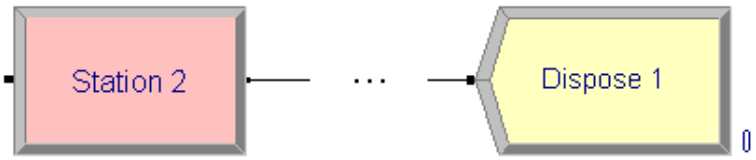
---

[22] Модули этого шаблона отличаются цветом в зависимости от назначения: работа со станциями – розовый, с конвейерами – зеленый, с транспортерами – голубой.

[23] Не следует путать сам модуль Station и соответствующий элемент модели. Модуль Station – это элемент графа модели, очередной шаг на пути транзакта. Модуль вызывает перемещение транзакта между станциями.

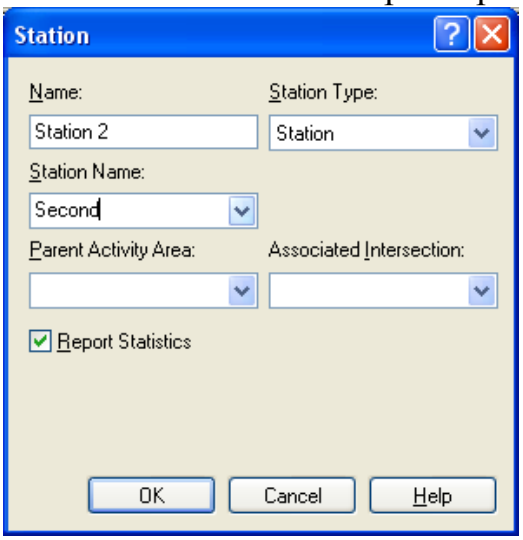


Есть еще один вариант использования модуля Station. Он может стоять первым в сегменте, как показано на рисунке.

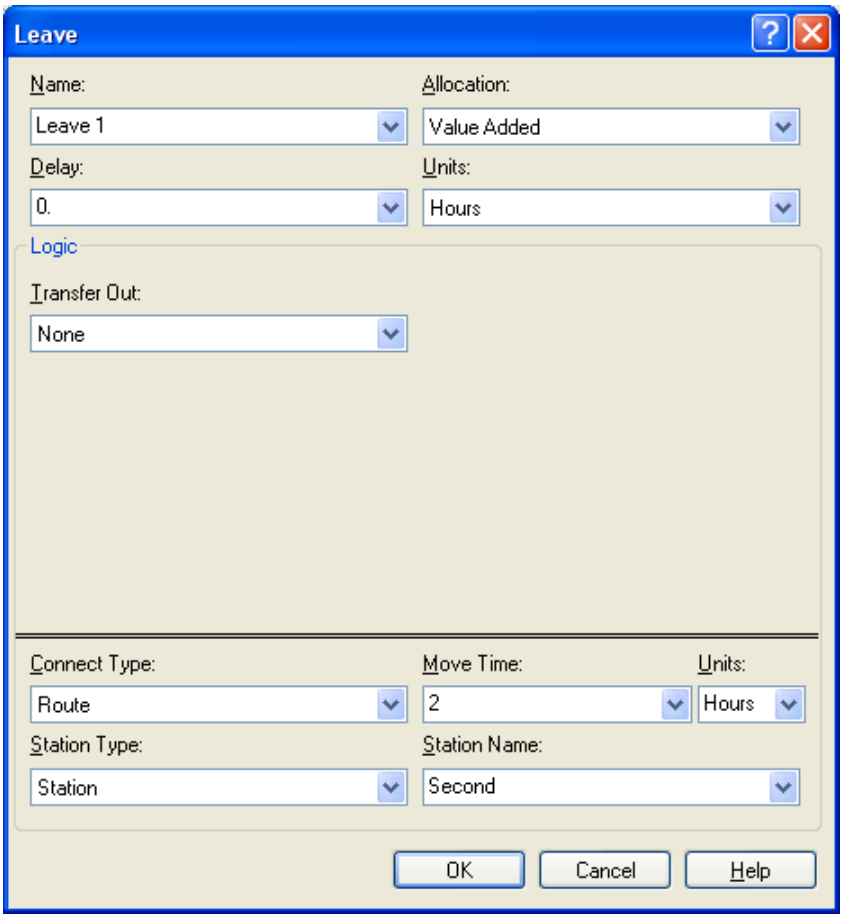


В приведенной модели в модуле Station 1 транзакт попадает на станцию «First» и после какой-то обработки, помощью модуля Leave 1 отправляется на станцию «Second». Когда транзакт прибудет на эту станцию, он окажется в модуле Station 2 и продолжит уже отсюда свое движение по модели.

Ниже показано окно параметров модуля Station 2.



Чтобы отправить транзакт со станции в конце сегмента можно применить один из нескольких модулей. В данном случае использован модуль **Leave**, его параметры показаны на рисунке.

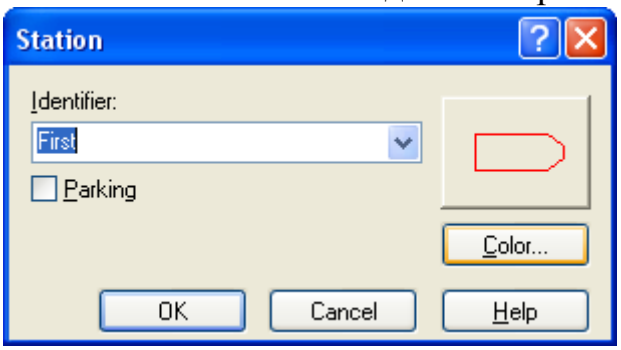


Мы видим, что модуль отправляет транзакт на станцию «Second», причем на перемещение необходимо 2 часа.

Существует несколько способов перемещения транзакта между станциями. В данном случае выбран вариант «Route» – путь. Перемещение по пути требует времени, но не предъявляет каких-либо дополнительных условий. Например, по пути в одно время может перемещаться сколько угодно транзактов.

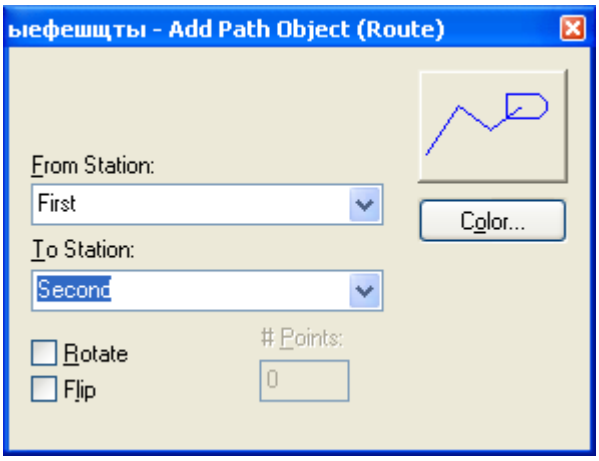
Использование станций может быть анимировано с помощью специального элемента анимации . Этот элемент расположен на панели инструментов Animate Transfer.

В окне элемента необходимо выбрать станцию:



Два графических элемента для двух станций показаны на рисунке.

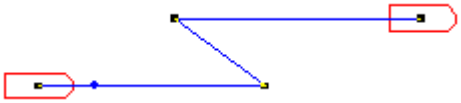
Для визуализации перемещения транзактов по пути есть элемент анимации  . В окне элемента необходимо выбрать начальную и конечную станции пути.



Выбрать две эти станции, если они уже размещены, можно и непосредственно на полотне:

Если станции еще не размещены на полотне, то после двойного щелчка мыши на пустом месте полотна возникнет элемент анимации станции.

Путь можно проложить так, как необходимо пользователю, введя несколько промежуточных точек одинарными щелчками мыши на полотне:



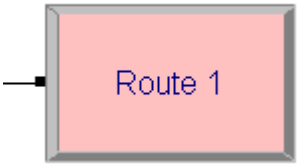
Путь от одной станции к другой и обратный путь это два разных пути, их нужно изображать отдельно.

Анимация пути приведена ниже. Сами станции во время анимации не отражаются (предполагается, что разработчик поместит на это место какой-нибудь символический рисунок), поэтому мы сделали надписи названий станций: First и Second. В данный момент по пути движутся три транзакта:

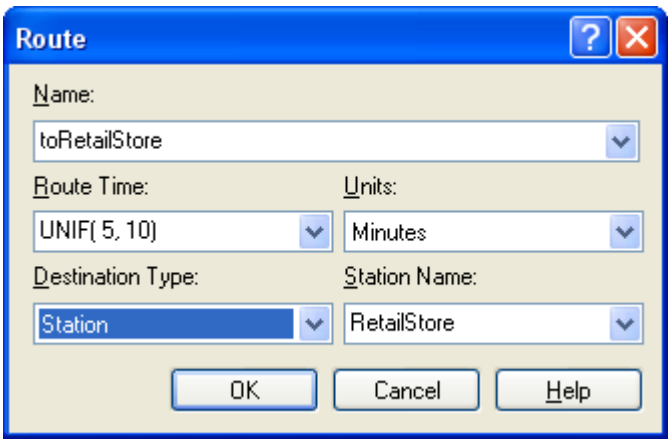
Рассмотрим теперь другие модули шаблона.

Модуль **Route** направляет транзакт на другую станцию по пути за определенное время. Располагается в конце сегмента модели.

Обозначение:

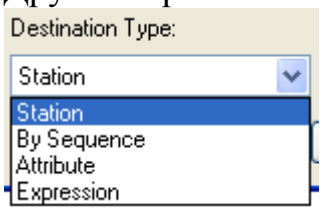


Параметры модуля Route приведены на рисунке. В параметрах модуля задается время необходимое для перехода (Route Time). Далее задается тип модуля. В простейшем случае, указывается конкретная станция (Station).



В данном случае, транзакт будет направлен на станцию «RetailStore» (розничный склад), причем путь займет от 5 до 10 минут.

Другие варианты назначения:



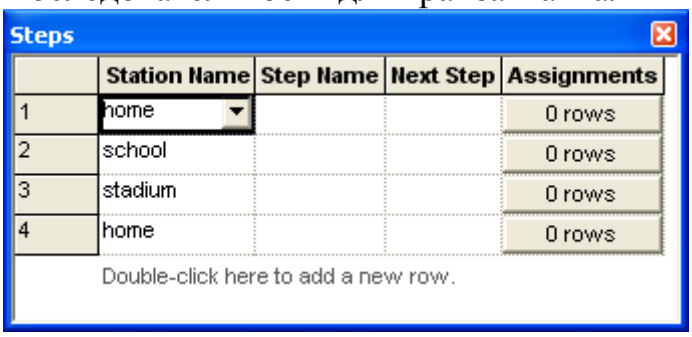
- *согласно последовательности* (by Sequence);
- *атрибут* (Attribute), транзакт будет направлен на станцию, имя которой сохранено в заданном атрибуте;
- *выражение* (Expression), станция определяется в произвольном выражении.

Очевидно, наибольший интерес представляет вариант с последовательностью (by Sequence). Arena предоставляет возможность создать некоторую **последовательность** станций, которые должен пройти транзакт (**Sequence**). Для разных транзактов можно задавать разные последовательности. Модуль Route автоматически определит следующую станцию для каждого транзакта.

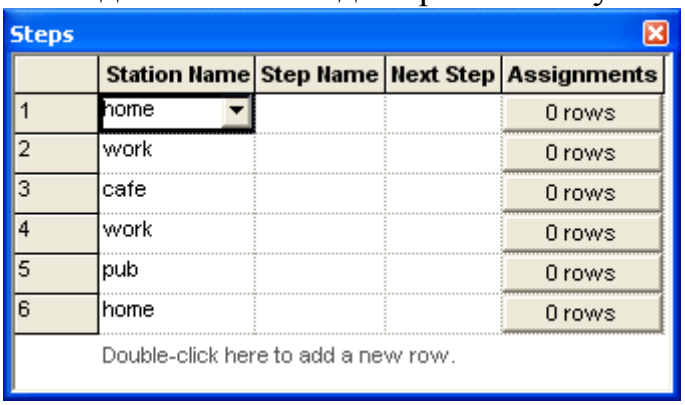
Чтобы создать последовательность используем невизуальный модуль Sequence вкладки Advanced Transfer. На рисунке приведены две последовательности для транзактов разных типов (для отца и для сына).

Колонка Name содержит идентификатор последовательности, колонка Steps (шаги), описывает станции, которые должны быть пройдены.

Последовательность для транзакта-мальчика:

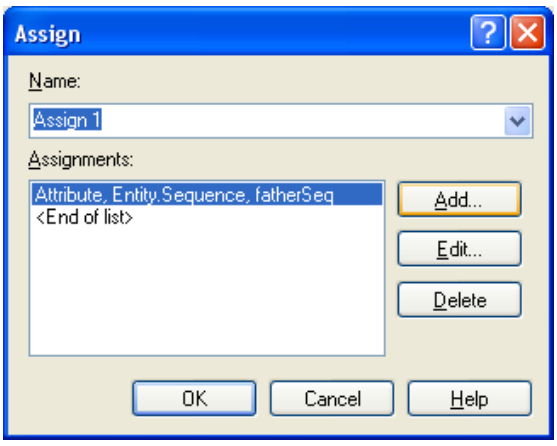


Последовательность для транзакта-мужчины:



Обратим внимание, одна и та же станция может входить в последовательность несколько раз. Так мужчина после обеда в кафе возвращается на работу.

Чтобы связать последовательность с транзактом необходимо использовать модуль Assign. Специальному атрибуту *Entity.Sequence* присваивается идентификатор последовательности, как показано на рисунке. Здесь транзакту, соответствующему мужчине устанавливается последовательность fatherSeq.

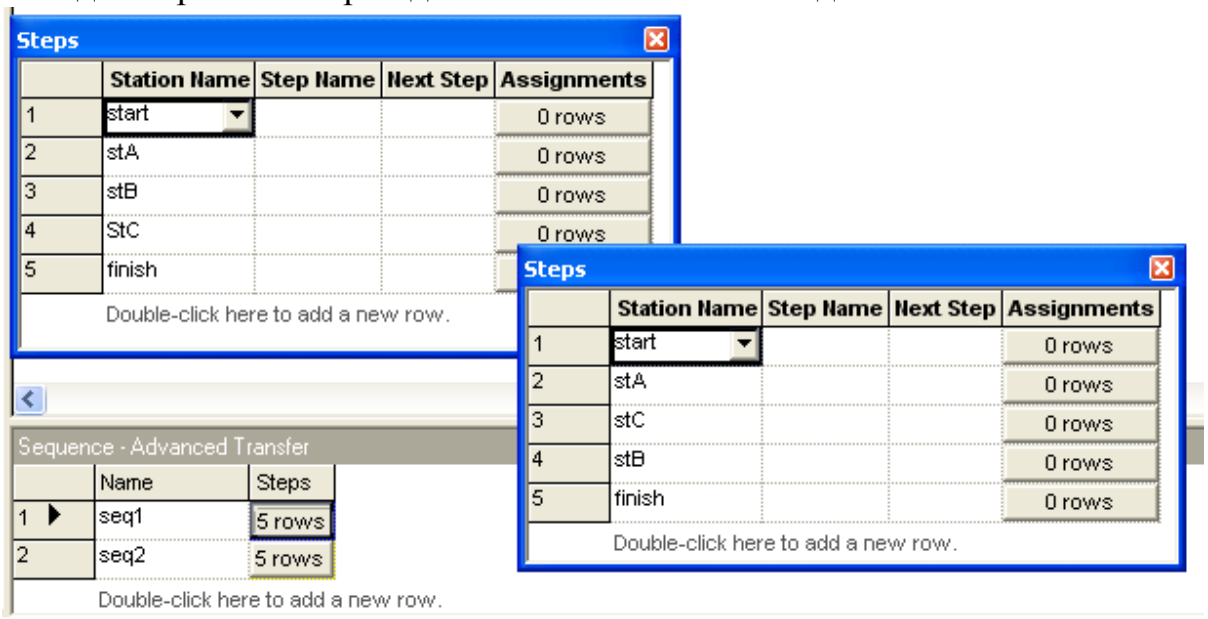


Таким образом, на каждой станции можно не думать о дальнейшем пути транзакта.

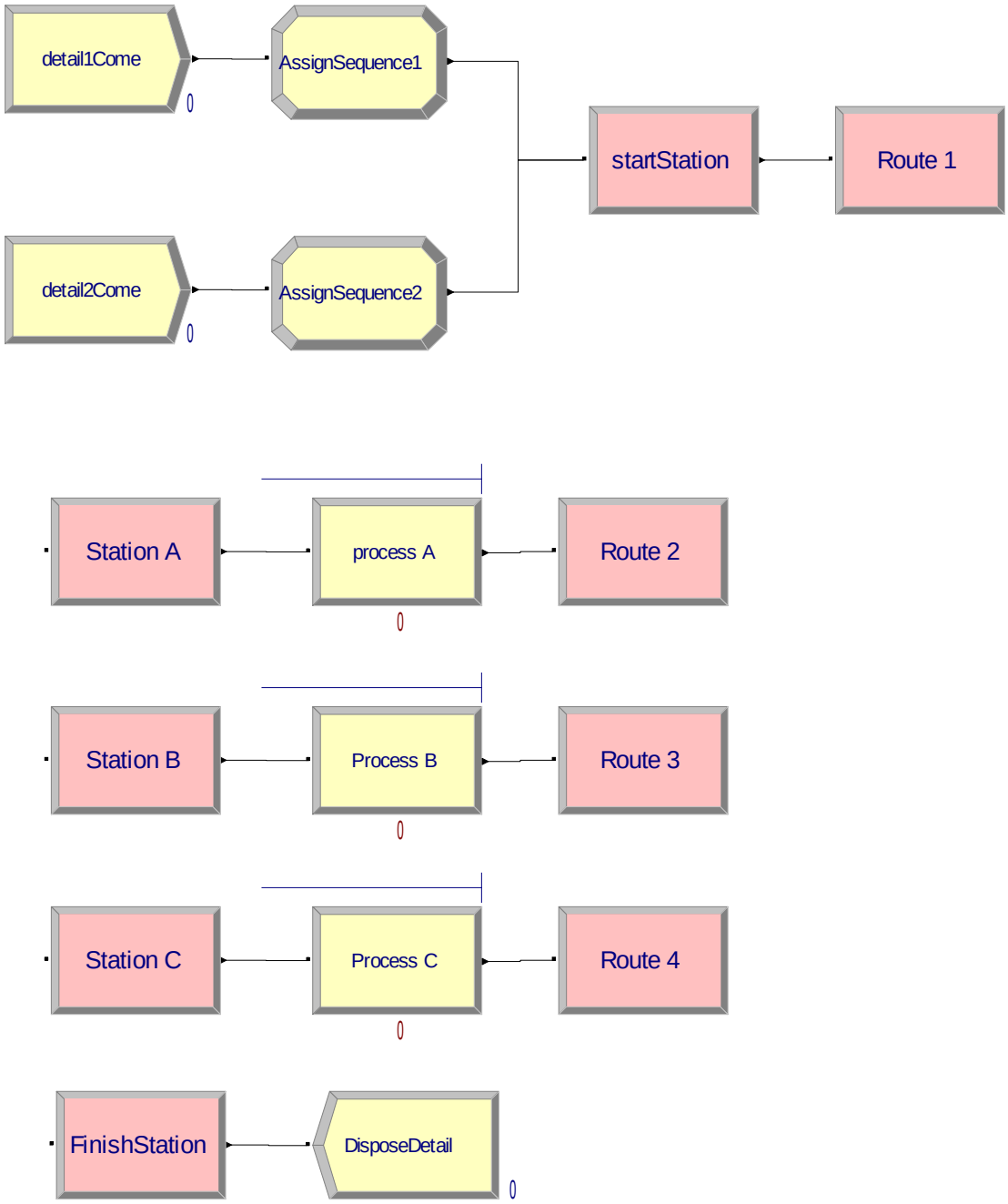
Последовательности позволяют осуществлять более тонкую настройку. В частности, с помощью колонки Assignments для очередного шага можно установить для транзакта значение атрибута со временем, необходимым для перехода на следующую станцию (очевидно, что переход «работа –> дом» потребует больше времени, чем переход «работа -> кафе»).

Пример 23. В цех поступает два типа деталей. Интервал между поступлениями одинаков и составляет 45 минут, закон распределения экспоненциальный. Деталь первого типа требует обработки на станках A, затем B, затем C. Вторая – на станках A -> C -> B. Время обработки детали на станке не зависит от типа детали и составляет 7±2 минуты на станке A, 5±2 на станке B, 10±3 минут на станке C. Закон распределения равномерный. Перемещение между станками занимает 2±1 минуты, закон распределения равномерный.

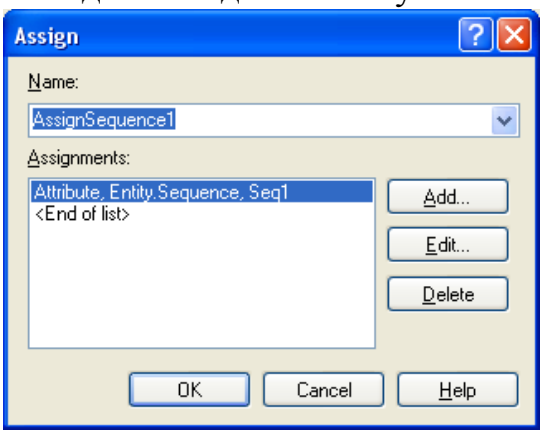
Реализовать такую модель просто. На рисунке показаны последовательности для первой и второй детали и шаги этих последовательностей.



Граф модели приведен на рисунке.

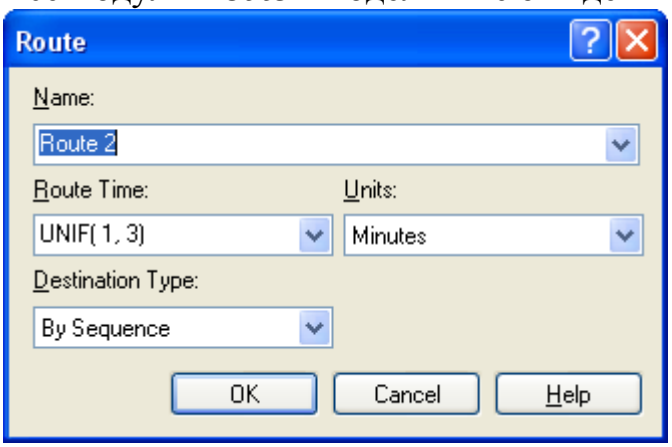


Два модуля Create создают транзакты-детали 2 типов. Затем в модулях Assign каждый тип детали получает последовательность.



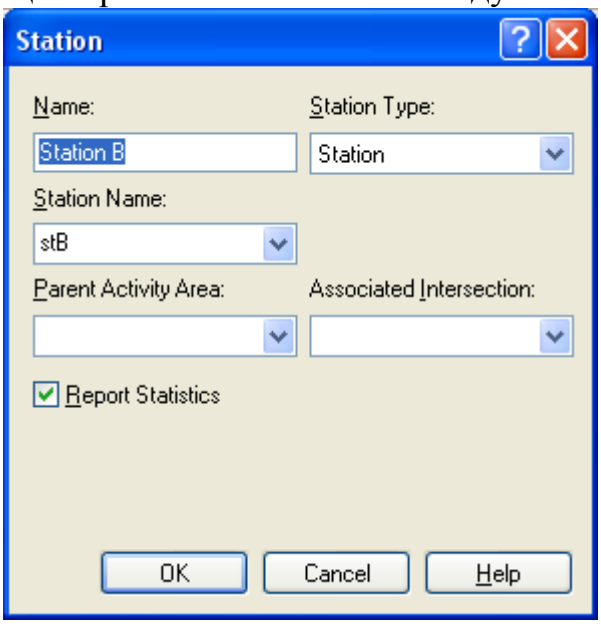
Транзакты поступают на станцию «start» с помощью модуля Station с именем startStation. Эта станция соответствует приемной зоне цеха. Далее с помощью модуля Route деталь передают в зону того станка, который должен ее обрабатывать.

Все модули Route в модели имеют идентичные параметры.



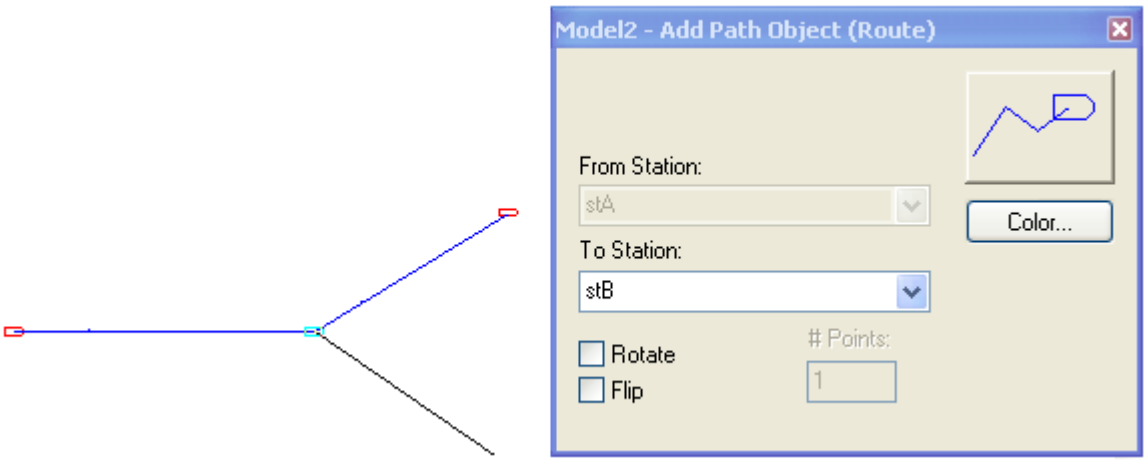
Деталь направляется к следующей станции в соответствии со своей последовательностью, причем переход занимает от 1 до 3 минут. Например, деталь 1 после станции «stB» отправится на станцию «stC», а деталь 2 с этой же станции «stB» отправится на станцию «finish».

Модули Station отвечают за появление деталей в зоне станка. Например, как только деталь будет доставлена к станку B (станция «stB») соответствующий транзакт возникнет в модуле Station B.

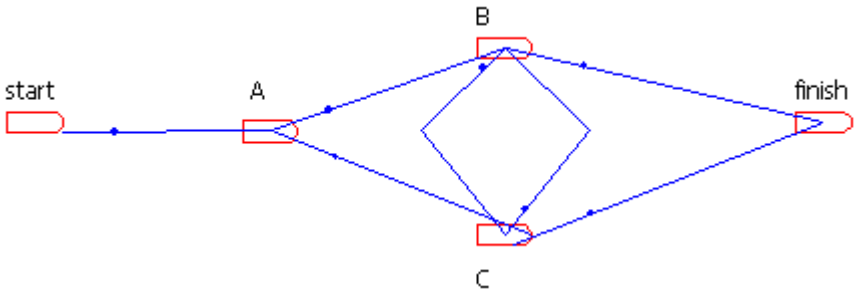


После того, как деталь попадет на станцию «finish» транзакт будет выведен из модели модулем Dispose.

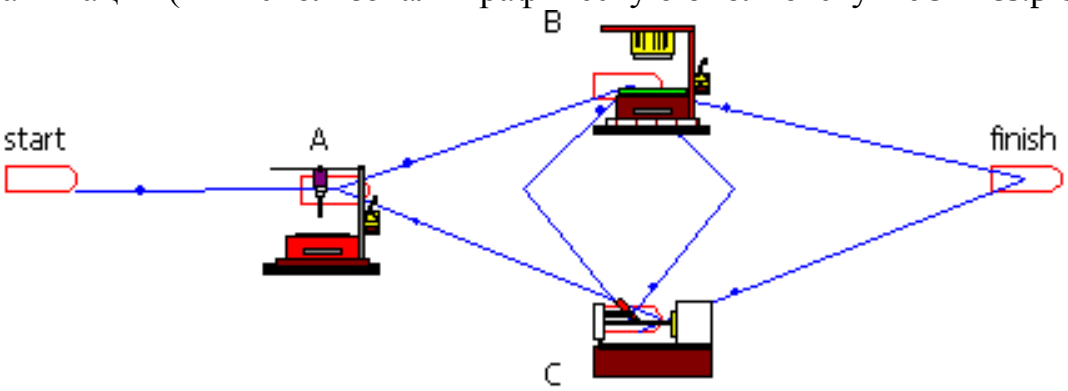
Для анимации станций и путей используем соответствующий элемент и прорисовываем все возможные пути.



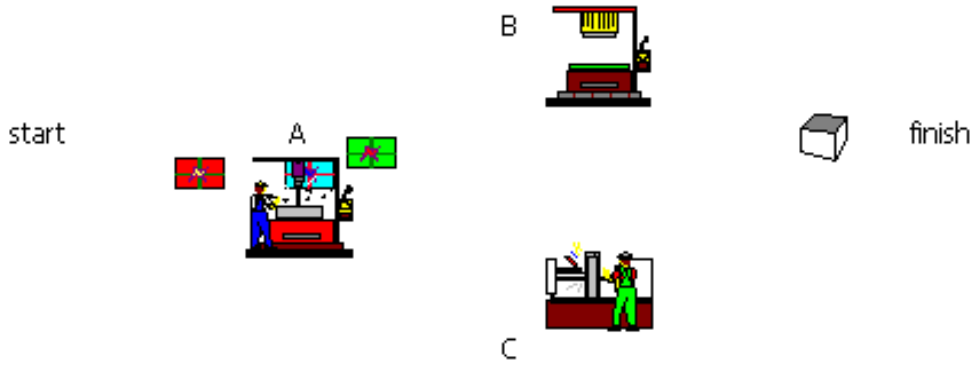
Всего 7 учатсков: start-stA, stA-stB, stA-stC, stB-stC, stC-stB, stB-finish, StC-finish.



Теперь разместим ресурсы-станки с помощью соответствующего элемента анимации (мы использовали графическую библиотеку machines.plb).



Работа модели показана на рисунке.



Модуль **Leave** – это универсальный модуль, соответствующий переходу от станции к станции. Он может заменить модуль Route. В соответствии с параметрами модуля, можно указать, что переход осуществляется с помощью пути (аналогично модулю Route) или с помощью транспортера, или конвейера или требует использования ресурса (например, лифта). Также моделируется необходимая задержка во времени перед отправкой.

Аналогично модуль **Enter** моделирует приход транзакта на станцию, обладает расширенными характеристиками и может заменить модуль Station в начале сегмента модели.

Контрольные вопросы

1) Что такое станция?
2) Зачем нужно использование станций?
3) Как можно визуализировать перемещение транзактов между станциями?
4) Что такое последовательность (Sequence) для модуля Route?
5) Как отразить в модели тот факт, что перемещение детали между цехами занимает 5±2 минут?
6) Как создать и использовать последовательность?
7) Что делает модуль Leave?
8) Что делает модуль Enter?
9) Сколько транзактов могут одновременно перемещаться по пути между двумя станциями?
10) Может ли время перемещения между станциями быть случайным?
11) Может ли транзакт перемещаться между станциями мгновенно?
12) Что делает модуль Station?

10 Проведение имитационного эксперимента в Арена

Рассмотрим пример проведения имитационного эксперимента над моделью, созданной в среде Arena.

Пример 24. Швейная фабрика имеет цех на 50 рабочих мест, которые оборудованы швейными машинками. Когда машинка выходит из строя она заменяется на резервную и передается в наладочный цех для ремонта [24].

Параметры процесса представлены в таблице.

Параметр модели	Значение
Срок безотказной работы машинки	157 часов, закон распределения – экспоненциальный
Время ремонта наладчиком швейной машинки	математическое ожидание 7, отклонение 1, закон распределения - нормальный
Число наладчиков	3, 4, 5 (чел)
Число резервных машинок	3, 4

Цель исследования состоит в следующем: определить оптимальный режим протекания процесса (число наладчиков и число резервных машинок).

Для принятия обоснованного решения необходимо оценить эффективность каждого варианта, сопоставив потери от простоя рабочих мест швейного цеха и затраты на заработную плату наладчиков и арендную плату за резервные машинки. Необходимо выбирать вариант, характеризующийся минимум суммарных затрат.

Данные для расчета затрат представлены в таблице.

Параметр	Обозначение	Значение
Зарплата наладчика	*z*	3.75 $ в час
Аренда машинки	*a*	25 $ в день
Убытки из-за простоя одного рабочего места	*v*	20 $ в час
Число наладчиков	*n*	-
Число швейных машинок	*m*	-
Нагрузка рабочих мест (процент времени, когда швеи заняты)	*Nag*	-

Суммарные затраты (C) определяем как:

$$C = C_a + C_z + C_v.$$

Сюда входят затраты на аренду машинок C_a, затраты на оплату труда наладчиков C_z и потери от простоя рабочих мест C_v. Все эти величины будем рассчитывать ориентируясь на один рабочий день – 8 часов.

Затраты на аренду машинок (C_a):

$$C_a = a * m$$

[24] Это классическая задача, рассматривается в книге Шрайбер Т. Дж. Моделирование на GPSS. — М.: Машиностроение, 1980. — 592 с.

Затраты на зарплату наладчикам (C_z):

$$C_z = z * 8 * n$$

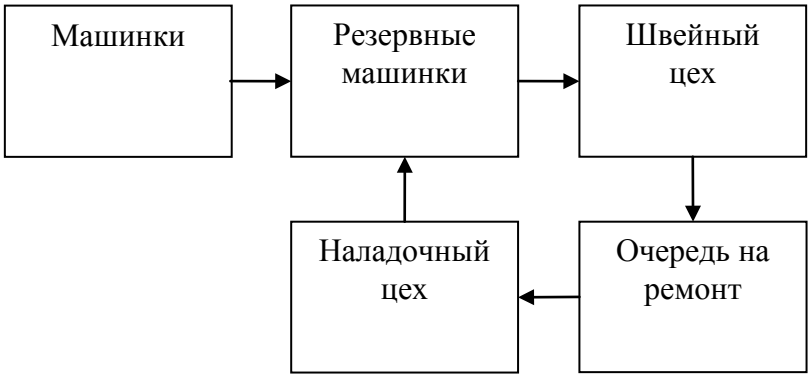
Убытки из-за простоев (C_v):

$$C_v = v * 8 * (1\text{-}Nag) * 50$$

Нам неизвестна только одна величина – процент простоя рабочих мест *(1-Nag)*. Чтобы определить *Nag*, загрузку рабочих мест для каждого из возможных вариантов мы можем использовать имитационную модель.

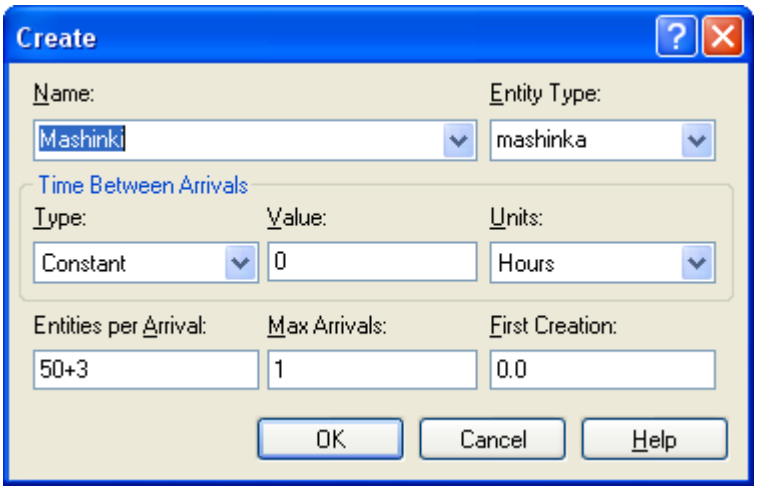
По своей природе имитационное моделирование не позволяет решать задачи оптимизации. Тем не менее, можно провести имитационный эксперимент, перебрав возможные варианты решения и выбрав наиболее эффективный.

Исследуемую систему можно представить в виде структурной схемы:

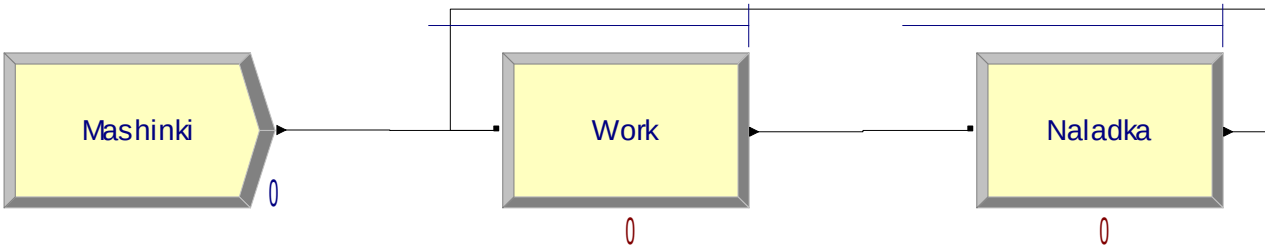


Реализуем модель в системе Arena.

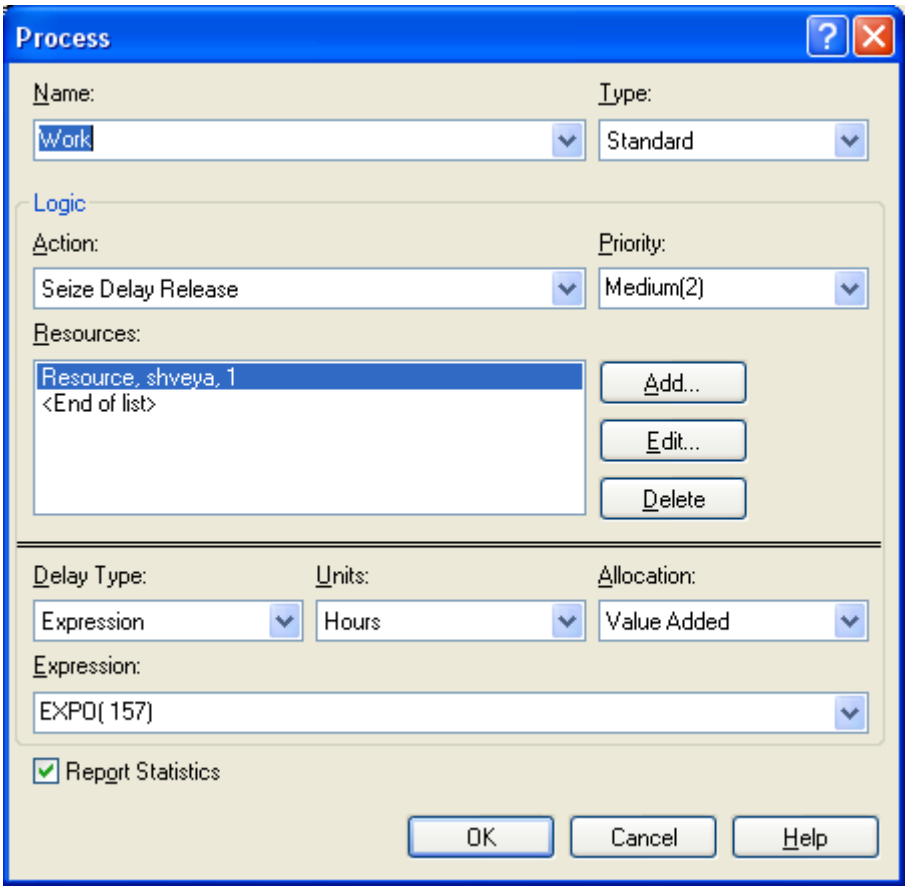
В данном примере транзактом будет швейная машинка. *50+m* швейных машинок будут введены в модель в момент времени 0, а затем будут циркулировать между модулями модели. То есть имеет место «замкнутая схема». Параметры модуля Create, который введет нужное число транзактов в модель, показаны на рисунке. На рисунке значение переменной *m* принято равным 3.



Граф модели показан на рисунке.

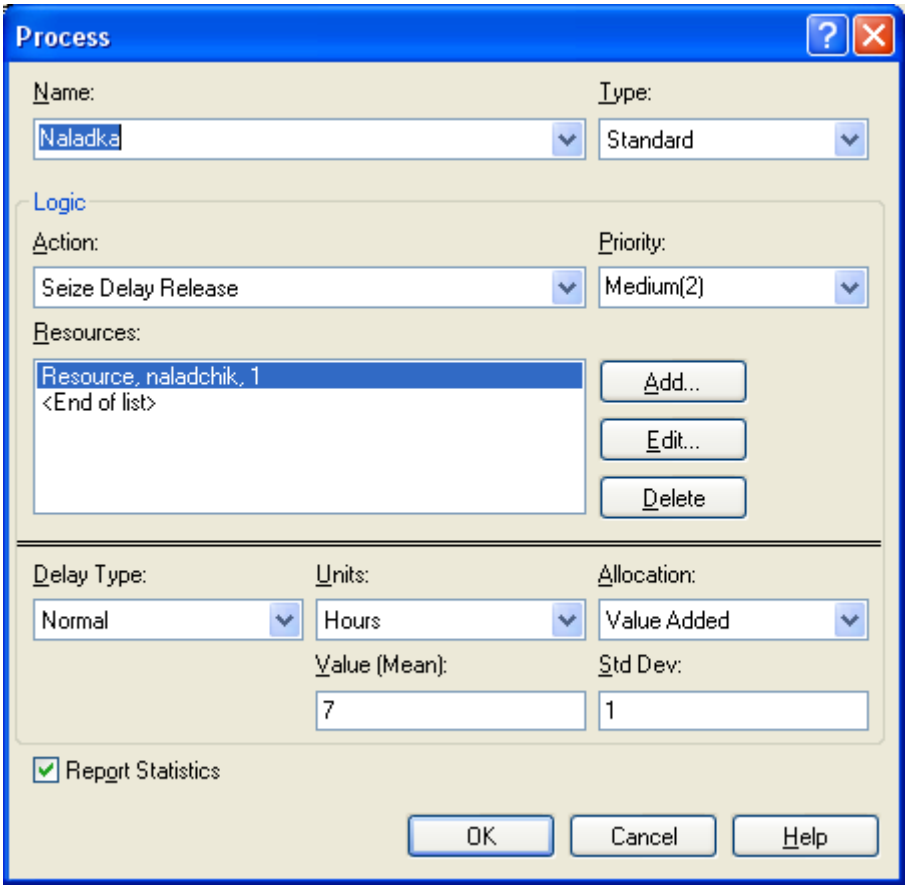


После создания транзакты-машинки отправляются в швейный цех, что представлено модулем типа Process с именем Work. «Обслуживание» машинки заключается в ее эксплуатации до тех пор, пока она не сломается. Одновременно может обслуживаться до 50 машинок. Используемый в ресурс – это швея.



Очередь модуля Work имеет смысл резерва, где находятся исправные машинки. Здесь они ожидают до тех пор, пока не потребуется заменить машинку, вышедшую их строя. Если машинка выйдет из строя, а резерв будет пуст – произойдет простой рабочего места.

Вышедшие из строя машинки направляются на ремонт, который описывается модулем Process с именем Naladka. Здесь может возникнуть очередь, если наладчики будут заняты.



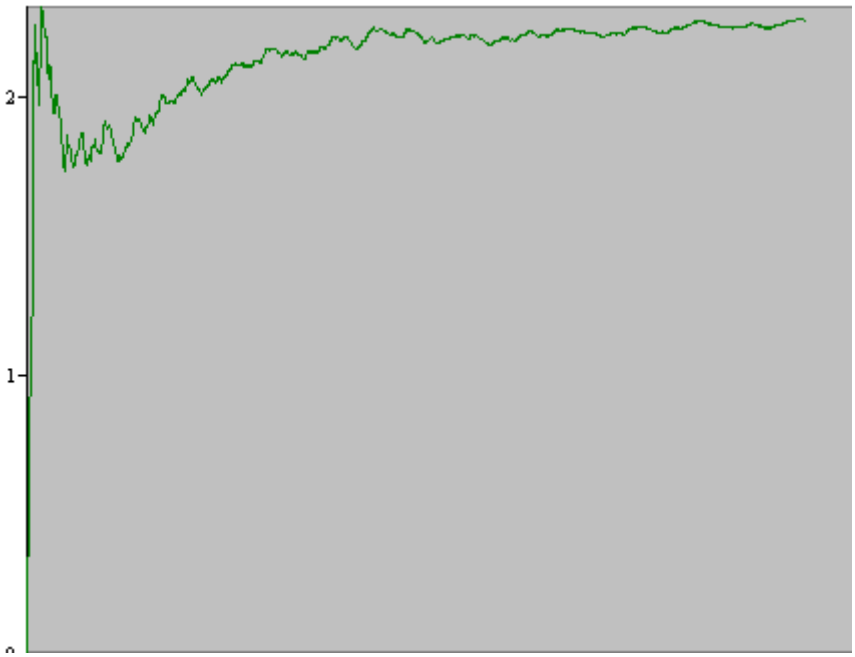
После окончания ремонта машинки возвращаются в швейный цех и помещаются в резерв.

Ресурсы модели:

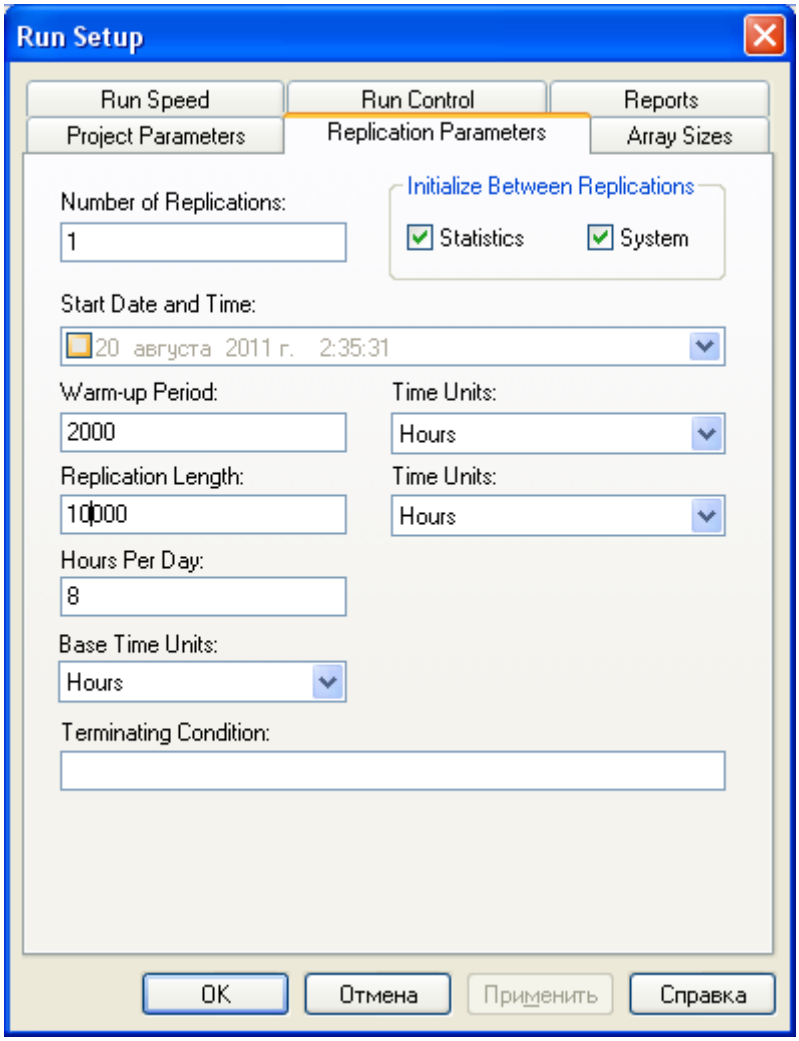
На рисунке показан график одного из параметров системы (в данном случае среднее число занятых наладчиков). Форма графика характерна для имитационного моделирования – имеется начальный период и стационарный период, когда колебания параметра заметно уменьшаются.

Причина заключается в том, что в начале моделирования в цех поступили новые машинки, имеется полный резерв, наладчики свободны, а очередь на наладку пуста.

Для проведения расчетов нас интересует только стационарный режим. В системе Arena можно задать так называемый разогревочный (Warm Up) период, статистика за который не собирается.



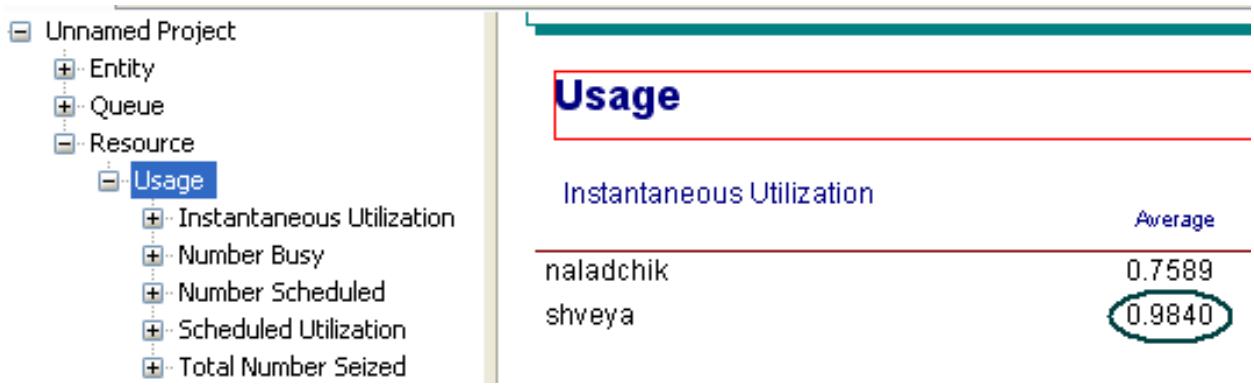
Укажем, длину периода моделирования 10000 часов, при разогревочном периоде 2000 часов.



Для моделирования при каждом варианте необходимо менять:

- мощность ресурса naladchik;
- число вводимых в модель резервных машинок в модуле Create.

В результате можно получить загрузку рабочих мест швей. Фрагмент отчета по результатам моделирования показан на рисунке.



В данном случае, загрузка составляет *Nag=0.984.*

На следующем этапе была проведена постановка имитационного эксперимента. Цель эксперимента – проследить поведение системы при изменении исходных данных и выбор оптимальных параметров - числа наладчиков (*n*) и числа резервных швейных машинок (*m*).

План и результаты эксперимента приведены в таблице.

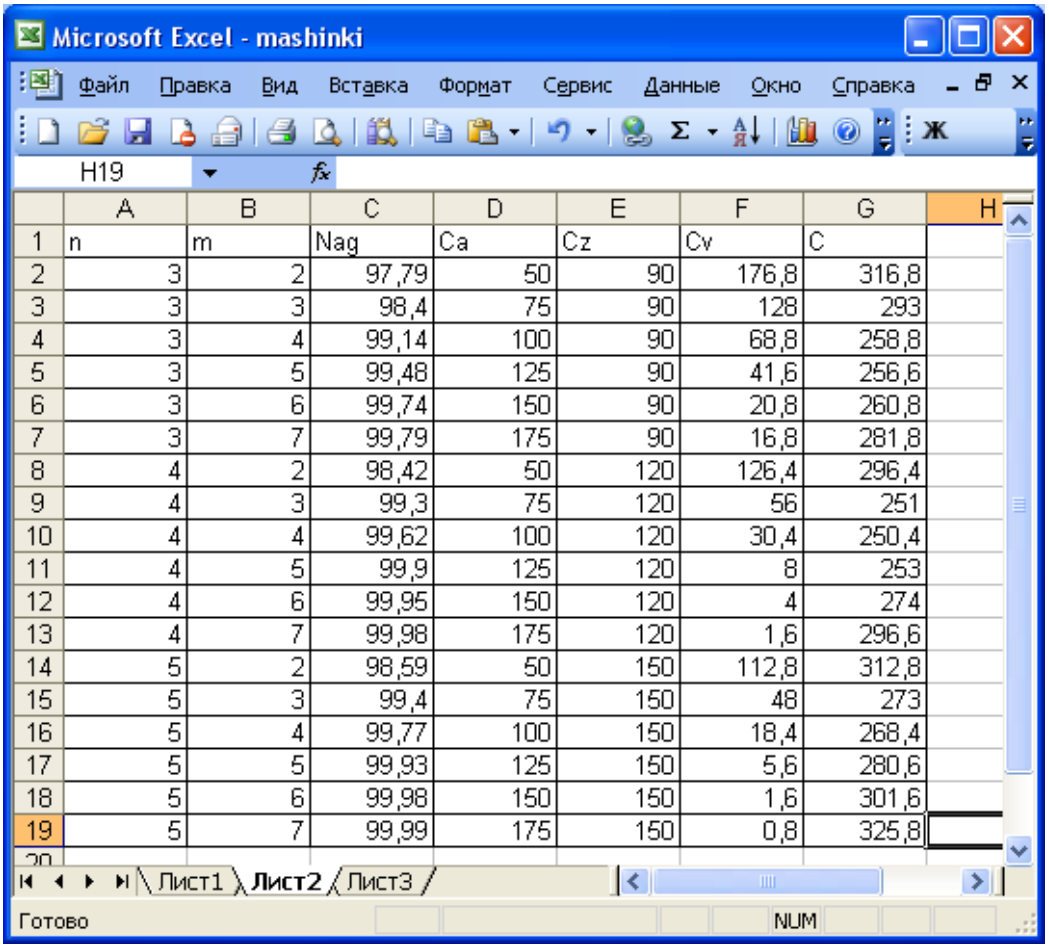
m \ n	3	4	5
2	97.79 %	98.42 %	98.59 %
3	98.40 %	99.30 %	99.40 %
4	99.14 %	99.62 %	99.77 %
5	99.48 %	99.90 %	99.93 %
6	99.74 %	99.95 %	99.98 %
7	99.79 %	99.98 %	99.99 %

Для проведения расчетов используем MS Excel.

По результатам эксперимента делаем вывод – оптимальным представляется вариант с использованием четырех наладчиков и четырех резервных швейных машинок [25].

Следующий пример относится к важному направлению – «управлению запасами». Разумеется, пример является максимально упрощенным, более реальные задачи имеют множество других деталей.

[25] Профессиональная версия Arena включает пакет OptQuest, позволяющий автоматически проводить имитационные эксперименты с моделями и искать оптимальные решения



Microsoft Excel - mashinki

Файл Правка Вид Вставка Формат Сервис Данные Окно Справка

H19

	A	B	C	D	E	F	G	H
1	n	m	Nag	Ca	Cz	Cv	C	
2	3	2	97,79	50	90	176,8	316,8	
3	3	3	98,4	75	90	128	293	
4	3	4	99,14	100	90	68,8	258,8	
5	3	5	99,48	125	90	41,6	256,6	
6	3	6	99,74	150	90	20,8	260,8	
7	3	7	99,79	175	90	16,8	281,8	
8	4	2	98,42	50	120	126,4	296,4	
9	4	3	99,3	75	120	56	251	
10	4	4	99,62	100	120	30,4	250,4	
11	4	5	99,9	125	120	8	253	
12	4	6	99,95	150	120	4	274	
13	4	7	99,98	175	120	1,6	296,6	
14	5	2	98,59	50	150	112,8	312,8	
15	5	3	99,4	75	150	48	273	
16	5	4	99,77	100	150	18,4	268,4	
17	5	5	99,93	125	150	5,6	280,6	
18	5	6	99,98	150	150	1,6	301,6	
19	5	7	99,99	175	150	0,8	325,8	

Лист1 Лист2 Лист3

Готово NUM

Пример 25. Предприятие реализует своим покупателям товары со склада. Покупатели появляются приблизительно каждый час, закон распределения экспоненциальный, размер покупки составляет от 1 до 5 единиц, закон распределения равномерный. Если на складе нет достаточного объема товара – клиент уходит. Максимальная вместимость склада – 25 единиц.

Как только уровень достигает некоторой критической точки [26] (будем считать таковым 3 единицы) выполняется звонок поставщику, который доставляет необходимое число единиц продукции - до 25 штук. При этом на заказ поставщику, сборы, доставку от поставщика, разгрузку и т.д. требуется от полутора до трех часов, с наиболее ожидаемым временем – 2 часа (будем считать закон распределения равномерным). После этого товар окажется на складе и будет готов к продаже.

Стоимостные параметры:

- выручка от продажи одной единицы товара за вычетом оптовой цены – 10$ (эта же сума выступает как потери от ухода клиента);
- расходы доставку заказам (также без оптовой цены товара) – 30$.

Необходимо определить оптимальную точку перезаказа.

Построим модель в системе Arena. Предлагаемый вариант не является единственно возможным. Есть и другие способы организации системы, например с помощью модуля Signal.

[26] В задачах управления запасами этот уровень носит название «точка перезаказа».

Прежде всего, введем переменные:

- *Level* – текущее число на складе. Начальное значение равно 25;

- *CriticalLevel* – точка перезаказа, уровень, при достижения которого делается заказ. Устанавливаем в 3, при проведении экспериментов меняем этот уровень;

- *Loses* – суммарные потери. Начальное значение равно 0;

- *OrderCosts* – затраты на обработку заказа, установлено значение 30.

Создание переменных в невизуальном модуле Variables показано на рисунке.

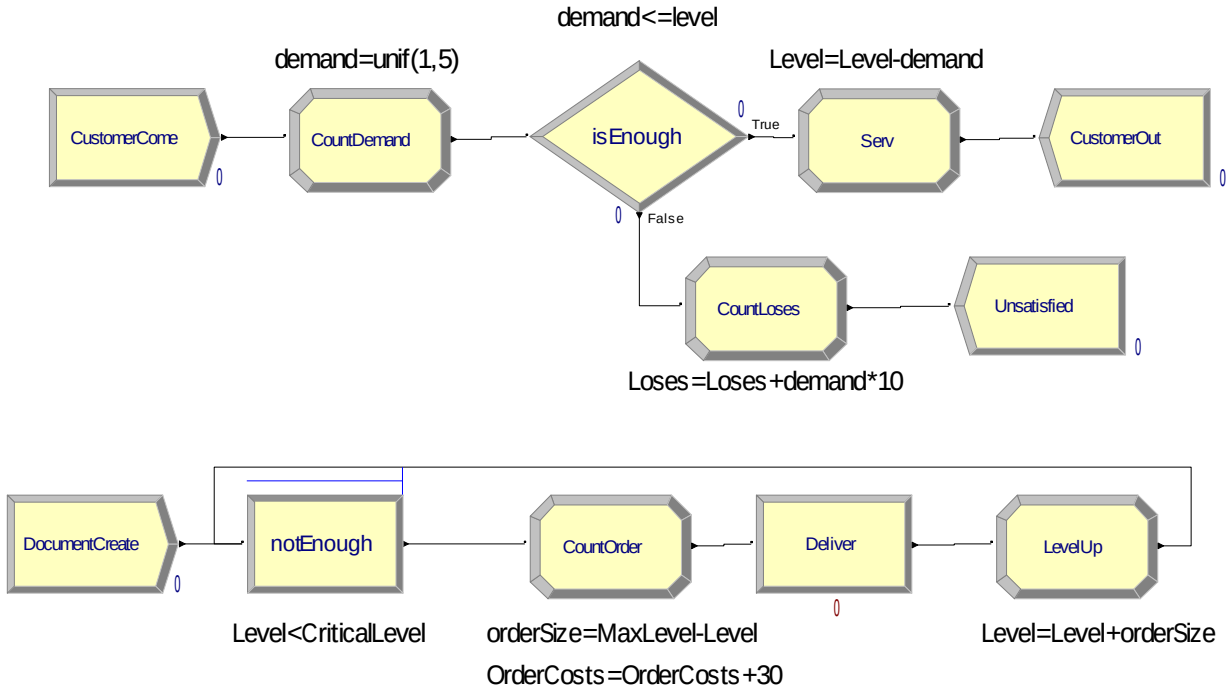
Variable - Basic Process

	Name	Rows	Columns	Data Type	Clear Option	File Name	Initial Values
1	Level			Real	System		1 rows
2	Loses			Real	System		1 rows
3	OrderCosts			Real	System		1 rows
4	CriticalLevel			Real	System		1 rows
5	MaxLevel			Real	System		1 rows

Double-click here to add a new row.

Для хранения информации о величине покупки каждого покупателя заведем у соответствующего транзакта атрибут *demand*. В случае ухода покупателя из-за отсутствия товара на складе потери составят: *$10*demand* (то есть 10 долларов за каждую единицу товара, который хотел приобрести клиент).

Граф модели показан на рисунке. Для наглядности все особенности расчетов показаны в виде комментариев возле блоков.



Верхний сегмент отвечает за покупателей. Покупатели вводятся в модель, далее в модуле CountDemand типа Assign рассчитывается потребность данного покупателя в товаре. Используется функция *expo()*, которая генерирует значение в соответствии с равномерным распределением и заданными максимальным и минимальным значением. В данном случае будет получено число от 1 до 5. Рассчитанное значение помещается в атрибут *demand*.

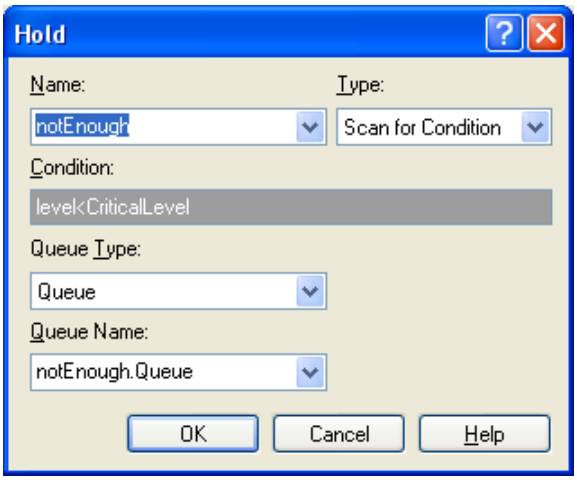
Если имеющийся запас не в состоянии удовлетворить потребность покупателя (проверяется в модуле IsEnough типа Decide), он уходит. Перед этим рассчитывается величина потерь – мы прибавляем к переменной *Loses* потенциальный чистый доход от ушедшего покупателя – размер его покупки, умноженный на 10.

В противном случае, заказ удовлетворяется, то есть уровень запаса в модуле CountLosses типа Assign снижается на объем требования покупателя. Затем транзакт-покупатель выводится из модели.

Нижний сегмент отвечает за заказы поставщику. Имеется специальный транзакт, который циркулирует по сегменту. Такой транзакт только один, он запускается в модель в момент времени 0 блоком Create.

Далее идет модуль типа Hold. Этот модуль заставляет транзакт ожидать момента заказа. Условием заказа поставщику является достижение критического уровня запаса. Транзакт последует дальше, когда будет выполнено условие *Level<CriticalLevel*.

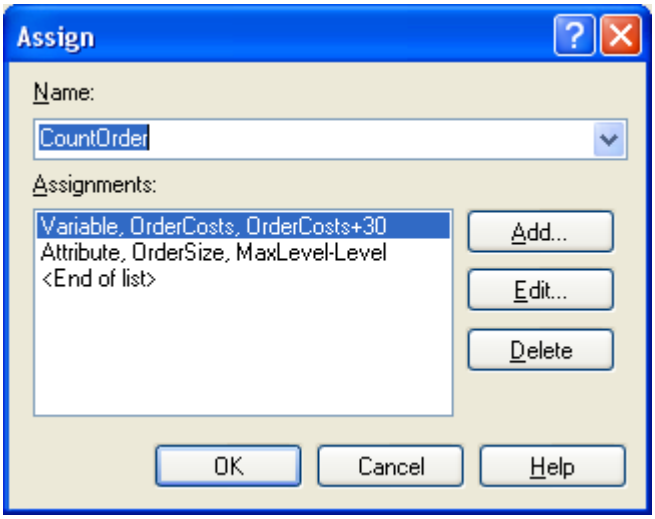
Параметры модуля Hold приведены на рисунке.



Далее в модуле CountOrder импа Assign рассчитывается необходимая величина поставки как максимальный уровень запаса минус текущий.Объем заказа будем хранить в атрибуте транзакта с именем OrderSize:

OrderSize = MaxLevel – Level.

В этом модуле также учитывается стоимость ее осуществления (к суммарным расходам на заказы добавляется $30).

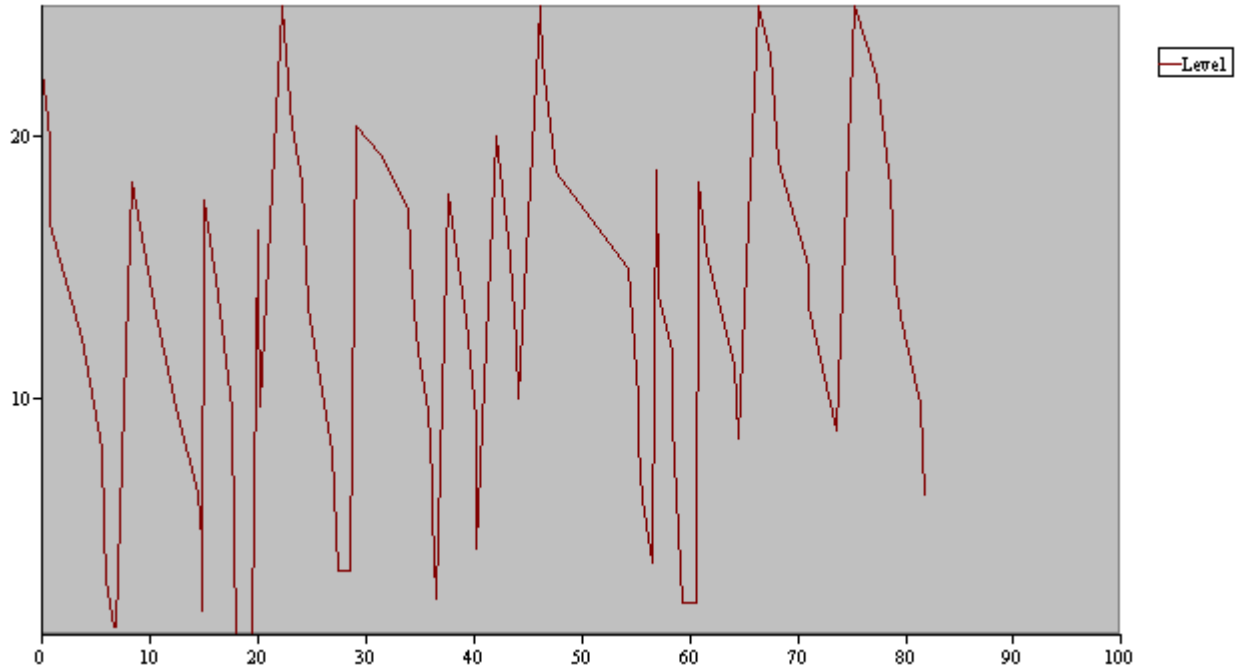


В модуле Deliver типа Process моделируется операция доставки. Выбран вариант модуля Delay – поскольку транзакт только один, нам нет необходимости моделировать ресурсы, например, грузовики на которых осуществляется доставка.

После того как товар доставлен и разгружен можно, рассчитать текущий уровень запаса с учетом объема поступившего заказа. Затем транзакт заказа вновь возвращается в модуль Hold для ожидания.

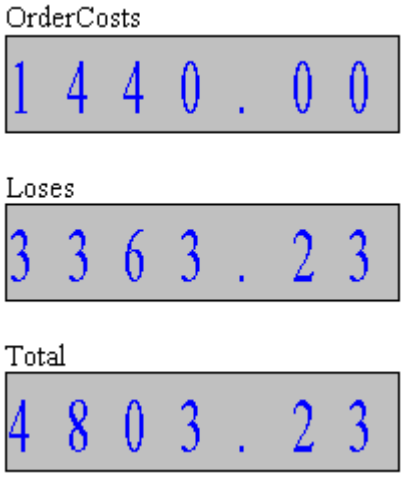
Проведем моделирование для 500 часов.

На рисунке приведен график переменной *Level*, текущего уровня запаса. График имеет традиционную для задач управления запасами форму «пилы».



Среди результатов моделирования будем отслеживать величину суммарных затрат, то есть значения переменных *Loses* и *OrderCosts*.

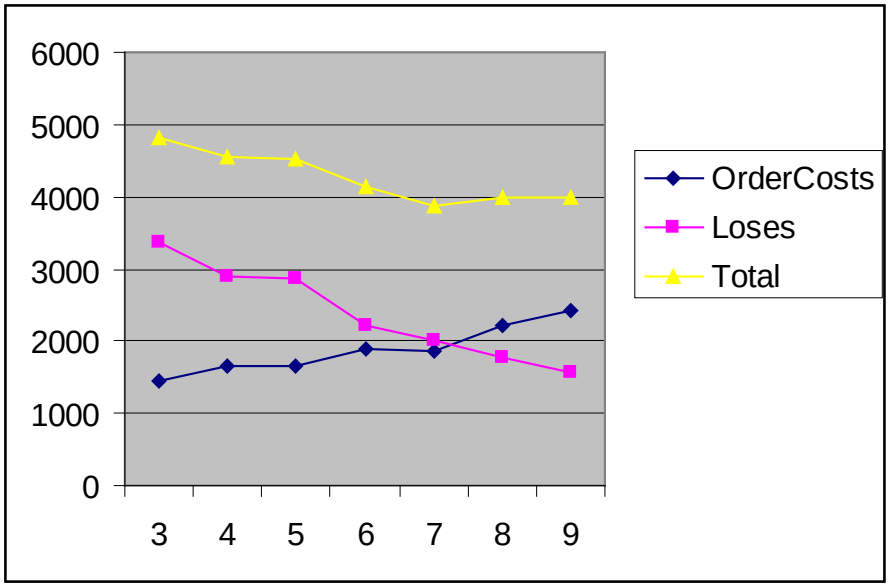
При критическом уровне в 3 единицы результат за 500 часов моделирования составит:



После семи экспериментов были получены следующие значения переменных:

CriticalLevel	OrderCosts	Loses	Total
3	1440	3363	4806
4	1650	2905	4559
5	1650	2878	4533
6	1890	2228	4124
7	1860	1996	3863
8	2220	1766	3994
9	2430	1564	4003

Результат может быть показан графически:



Мы видим, что по мере увеличения точки перезаказа растут затраты на обслуживание заказов, но уменьшаются потери от ушедших клиентов. Оптимальной точкой пополнения запасов является 7.

Задача 5. Дополним предыдущий пример (**Пример 24**). Каждые 8 часов поставщик осуществляет плановый завоз товара, причем стоимость такого завоза существенно ниже – $10. При этом необходимо определить потребность в товаре за 2 часа до начала поставки.

Необходимо продумать оптимальную стратегию закупок.

Задача 6. Провести моделирование процесса приема экзамена в вузе. Процедура приема стандартная – первая группа студентов (5 человек) появляется в аудитории в 9:00. Каждый студент подходит к преподавателю, берет билет, начинает подготовку. Далее студент отвечает на билет, возможно, получает положительную либо отрицательную оценку. Также возможен вариант, когда преподаватель задаст одно или несколько дополнительных заданий, перед ответом на которые студент также может подготовиться. После того, как студент покинул аудиторию, следующий студент может подойти, взять билет, начать подготовку. Экзамен будет закончен, когда последний студент получит оценку.

Следует учесть, что число студентов в группе ограничено.Также следует иметь в виду, что если к преподавателю обращаются несколько студентов, то приоритет должен быть отдан тому студенту, который должен получить билет. Вероятно, стоит также предусмотреть возможность того, что студент будет не готов к экзамену и покинет аудитория сразу после получения билета. Возможно, кто-то из студентов станет отвечать без подготовки.

Время на разные операции, а также различные детали модели взять из собственного опыта.

Необходимо сделать анимацию, которая покажет студентов за подготовкой, студентов в аудитории, студентов вне аудитории, преподавателя и т.д. Установить часы для отражения текущего времени.

Основная цель – закончить экзамен как можно раньше. Также следует обеспечить, чтобы студенты не ждали лишнее время. Нужно составить схему прихода студентов, преподаватель может огласить график накануне, во время консультации. Возможно, следует организовать приход несколькими группами или каким-то еще способом.

Рассмотреть разные варианты, оценить время ожидания студентов и время окончания экзамена преподавателем. Провести другие исследования модели для сокращения времени.

Заключение

В учебном пособии были рассмотрены вопросы имитационного моделирования экономических систем с использованием системы имитационного моделирования Rockwell Arena.

Содержание книги преимущественно раскрывает технологическую сторону построения имитационных моделей. Рассмотренный материал и приведенные примеры позволяют студенту научиться строить модели, отражающие достаточно сложные процессы в управлении предприятиями, эффективно использовать разнообразные компоненты системы имитационного моделирования.

Однако владение инструментом – это только часть знаний и навыков специалиста. Не менее важным является освоение содержательной стороны имитационного моделирования. Построение модели должно иметь ясную цель; структура модель должна соответствовать изучаемому процессу; следует корректно собрать и правильно обработать исходные данные; нужно спланировать и поставить эксперимент; модель необходимо использовать для формирования и оценки конкретных управленческих решений.

В дальнейшем освоении методологии имитационного моделирования поможет обращение к учебникам, приведенным в списке основной литературы, а также практика построения и исследования имитационных моделей.

Литература

Основная

1. Лоу А., Кельтон В. Имитационное моделирование. – СПб: Питер, 2004. – 848 с.
2. Шеннон Р. Имитационное моделирование систем – искусство и наука. – М.: Мир, 1978. – 417 с.

Дополнительная

3. Емельянов А. А., Власова Е. А., Дума Р. В. Имитационное моделирование экономических процессов. – М.: Ф и С, 2009. – 416 с.
4. Форрестер Дж. Основы кибернетики предприятия (индустриальная динамика) / пер. с англ., общая редакция Д.М. Гвишиани. – М.: Прогресс, 1971. – 340 с.
5. Simulation with Arena, 5th Edition by W. David Kelton, Randall P. Sadowski, Nancy B. Swets Rockwell Automation McGraw-Hill, 2010.
6. Altiok T., Melamed B. Simulation Modeling and Analysis with ARENA. – Academic Press, 2007. - 456.

Ресурсы в сети Интернет

7. www.arenasimulation.com – официальный сайт системы Rockwell Arena.
8. www.gpss.ru – портал имитационного моделирования. Материалы всероссийских конференций «ИММОД».
9. www.wintersim.org – Материалы ежегодных международных конференций «Winter Simulation Conference».

Вопросы к экзамену

по дисциплине «Имитационное моделирование экономических процессов»

1. Понятия системы и модели
2. Особенности сложных социально-экономических систем
3. Классификация моделей
4. Требования к моделям
5. Метод имитационного моделирования. Возможности, преимущества, области применения
6. Основные классы имитационных моделей
7. Непрерывное имитационное моделирование
8. Этапы имитационного моделирования
9. Подготовительные этапы имитационного моделирования
10. Этапы построения и проверки моделей
11. Стратегическое и тактическое планирование имитационного эксперимента
12. Планирование имитационных экспериментов по схеме полного факторного эксперимента
13. Учет модельного времени
14. Цепи текущих событий и будущих событий
15. Генерация случайных чисел. Базовый датчик. Требования к базовым датчикам
16. Моделирование случайных событий, дискретных и непрерывных случайных величин, заданных таблично
17. Система Ithink. Возможности и принципы работы
18. Система имитационного моделирования GPSS. Возможности и принципы работы
19. Система имитационного моделирования Arena. Возможности и принципы работы
20. Система имитационного моделирования Arena. Моделирование ресурсов и очередей
21. Система имитационного моделирования Arena. Пулы ресурсов, расписания
22. Система имитационного моделирования Arena. Модули шаблона «Basic process»
23. Система имитационного моделирования Arena. Транзакты, атрибуты, переменные
24. Система имитационного моделирования Arena. Моделирование процессов сборки и разделения. Модули Batch, Separate
25. Система имитационного моделирования Arena. Задержка и синхронизация транзактов. Модули Hold, Match
26. Система имитационного моделирования Arena. Моделирование процессов с использовании станций
27. Система имитационного моделирования Arena. Средства анимации