Проектные задания по имитационному моделированию в среде AnyLogic

Автор проектных заданий:
Переслегина Ольга Константиновна

Москва – 2016 год

Агентная модель работы аэропорта.

Директор аэропорта решил выявить недостатки работы своего аэропорта, которые могут возникнуть в процессе обслуживания пассажиров перед вылетом, с целью повысить уровень предоставления услуг. При этом известно, что:

- Пассажиры прибывают в среднем за 2,5-2 часа до вылета самолёта (то есть подавляющая часть пассажиров прибывает в аэропорт в течение получаса);
- Среднее число пассажиров одного самолёта – 500;
- Пассажиры обязаны пройти процедуру регистрации (но мы учитываем, что некоторые пассажиры могли зарегистрироваться на рейс дома в режиме on-line, что исключает необходимость проходить регистрацию непосредственно в аэропорту);
- Все пассажиры обязаны пройти паспортный контроль и досмотр вещей (как багажа, так и ручной клади);
- Пассажиры обязаны сдать багаж, не относящийся к ручной клади (при этом, не все пассажиры имеют при себе багаж);
- Перелёты совершаются внутри страны (что исключает необходимость проводить таможенный контроль).

Задача. Построить модель прохождения всех инстанций аэропорта перед вылетом с целью выявления недостатков обслуживания и их дальнейшего устранения.

Для удобства будем считать, что никто из пассажиров не «промахнулся» с весом багажа (так как для его веса есть ограничения), не забыли дома паспорта и прочие нужные вещи. В контексте данной модели такие события не являются важными, поэтому мы их не рассматриваем.

Создадим чистую модель (Файл → Создать → Модель), на втором этапе выберем "с нуля"(рисунки 1 и 2).

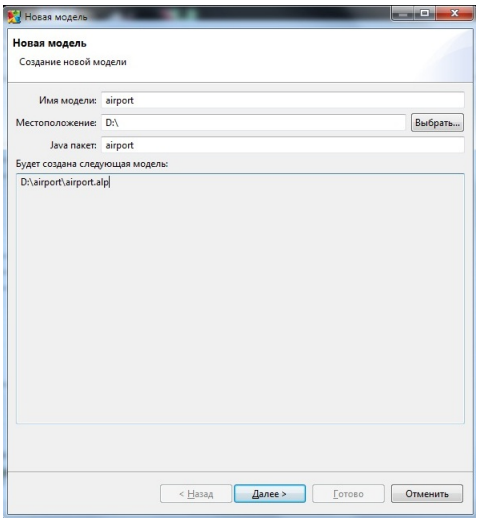

Рис. 1: Создание модели

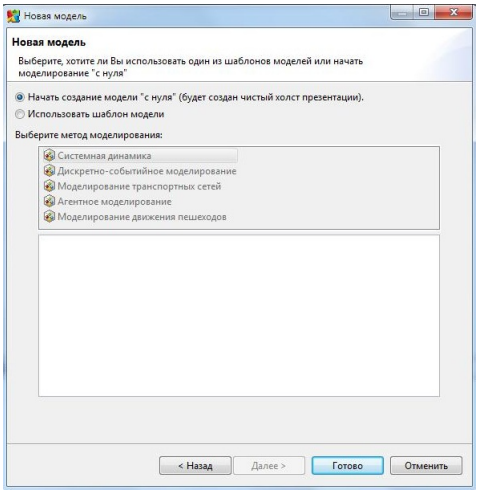

Рис. 2: Выбор пустого шаблона

Для нашей агентной модели воспользуемся объектами с закладки Пешеходная библиотека (рисунок 3).

Рис. 3: Палитра «Пешеходная библиотека»

Прежде чем начать строить нашу модель, мы сделаем небольшую схему аэропорта для наглядности. Для этого с закладки Презентация воспользуемся следующими инструментами: Линия, Ломаная, Прямоугольник. Для того чтобы воспользоваться инструментом, необходимо дважды щелкнуть на его название (или один раз на значок карандаша, находящийся напротив соответствующего инструмента). Также мы будем использовать Текст, но для его использования необходимо «захватить» его и перенести в необходимую точку рабочего места.

Начнём с постройки стен аэропорта. Для этого выберем инструмент Ломаная и изобразим ломаную линию, как показано на рисунке (4). Для того, чтобы данная ломаная служила в нашей модели стеной, нам нужно на ней нажать выбрать «Группировка → Создать группу» и дать имя созданной группе wall.

Затем подпишем с помощью инструмента Текст названия инстанций на том месте, где они будут находиться. Их 4: **registration** (регистрация), **control** (паспортный контроль), **inspection** (досмотр) и **baggage** (сдача багажа). Вместе с тем разместим рядом с надписями **control** и **baggage** прямоугольники (рисунок 5).

Но одних прямоугольных зон нам будет недостаточно, поэтому внутри каждого прямоугольника построим по одной линии (рисунки 6 и 7).

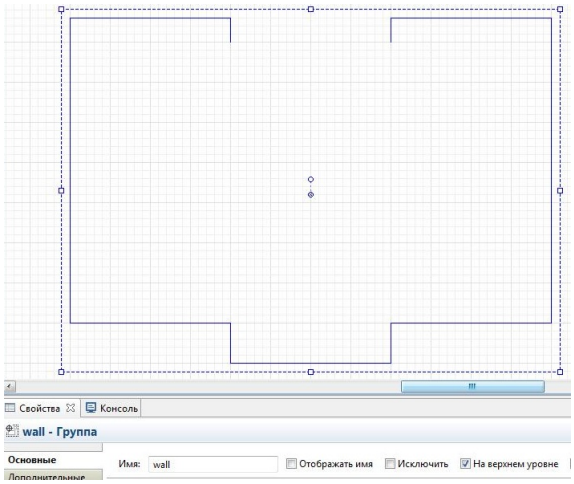

Рис. 4: Стены аэропорта

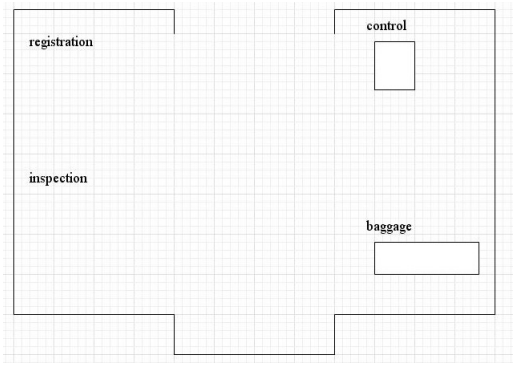

Рис. 5: Обозначение необходимых инстанций

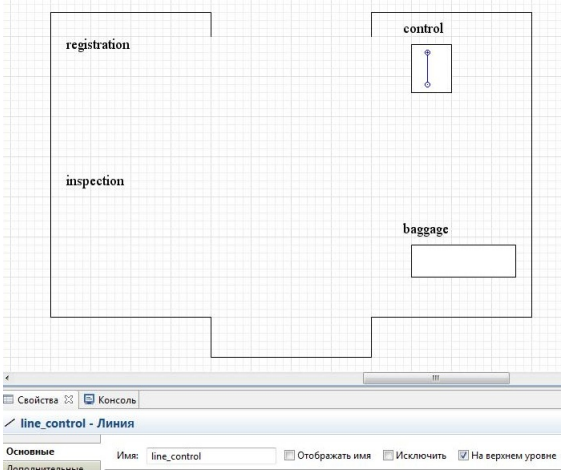

Рис. 6: Построение линии инстанции **control**

Далее поместим линию, где будут появляться наши агенты. Назовём её **input** (рисунок 8).

Ещё нам потребуется линия выхода. Её мы назовём **output** (рисунок 9).

Теперь нам необходимо разобраться с инстанциями **registration** и **inspection**. Здесь нам надо построить по два столбца, состоящих из трёх линий, расположенных друг над другом, как показано на рисунке 10:

Затем объединим их в четыре группы: группы **group_reg** и **group_insp** отвечают за стойки обслуживания; **queue_reg** и **queue_insp** – за очереди к ним. Для объединения линий в группы необходимо, с зажатой клавишей Shift, последовательно выделить все линии, относящиеся к одной группе, кликнуть правой кнопкой мыши, выбрать «Группировка → Создать группу» и дать имя созданной группе, как показано на рисунках 11 и 12.

Аналогично необходимо создать группы для инстанции **inspection** (рисунки 13 и 14).

Итак, схема нашего аэропорта готова. Теперь приступим к описанию модели. Для начала построим цепочку, как показано на рисунке 15

Для её построения были использованы следующие объекты пешеходной библиотеки:

– **pedSourse** – начальная точка нашей схемы, здесь создаются пешеходы;

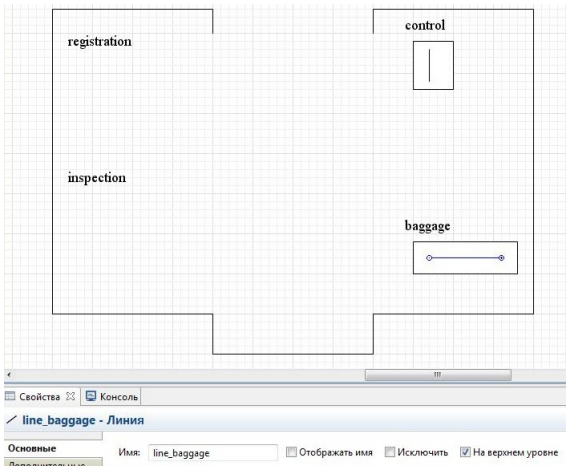

Рис. 7: Построение линии инстанции **baggage**

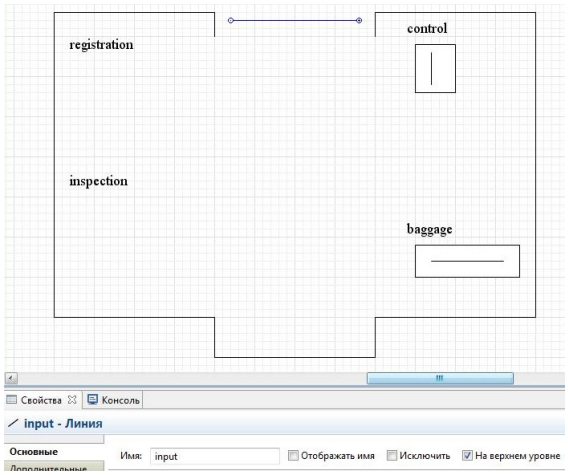

Рис. 8: Место появления агентов

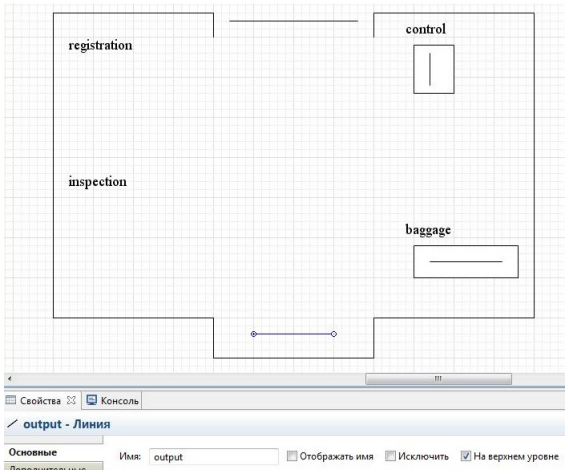

Рис. 9: Место выхода агентов из модели

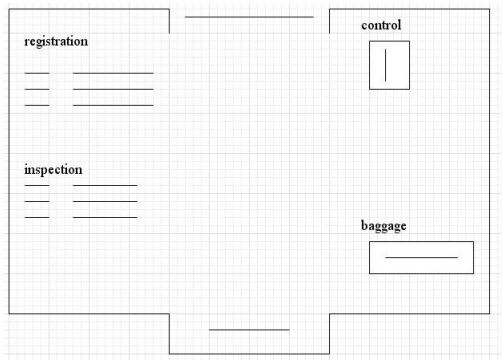

Рис. 10: Построение прямых

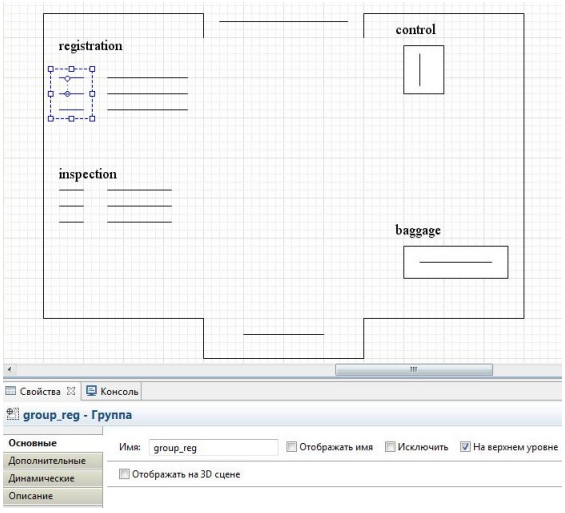

Рис. 11: Создание группы линий **group_reg**

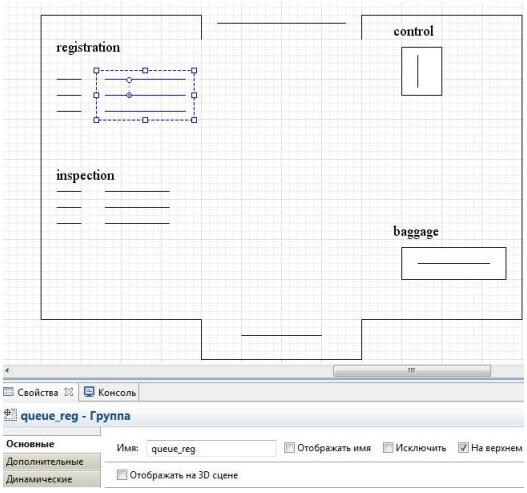

Рис. 12: Создание группы линий **queue_reg**

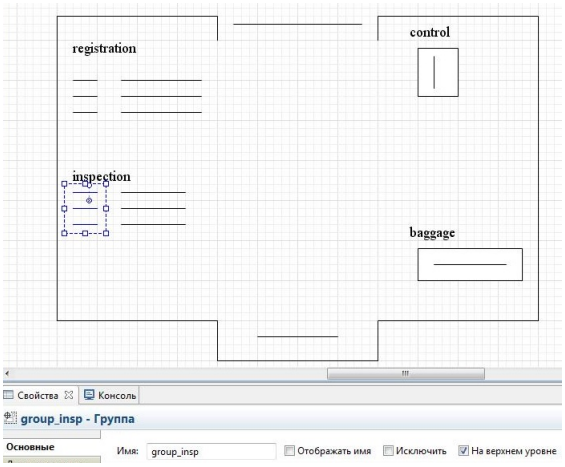

Рис. 13: Создание группы линий **group_insp**

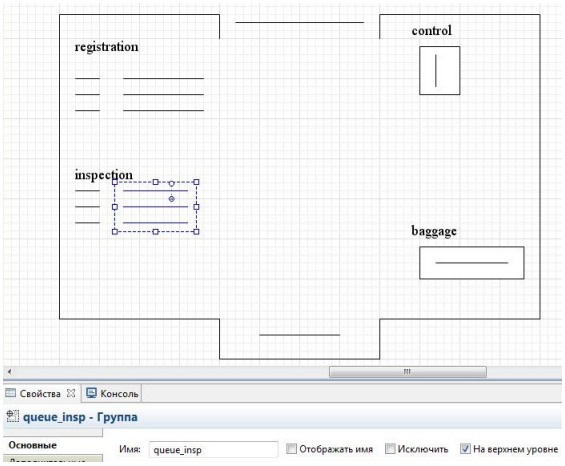

Рис. 14: Создание группы линий **queue_insp**

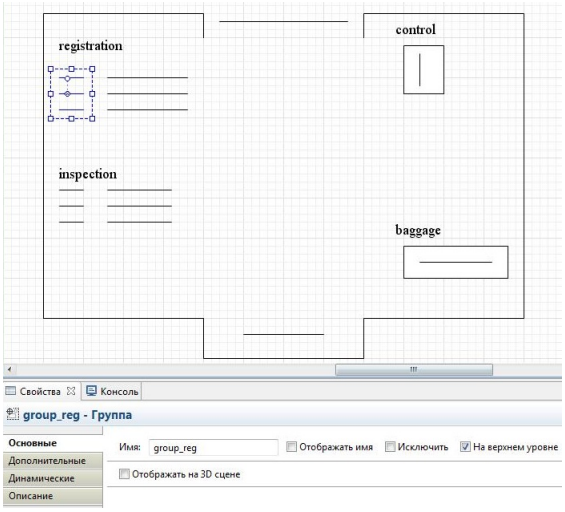

Рис. 15: Построение блок-схемы модели

– **pedService** – создаёт очередь;

– **pedGoTo** – задаёт направление движения;

– **pedWait** – ожидание (задержка в заданной точке);

– **pedSelectOutput** – здесь происходит выбор направления движения в зависимости от коэффициента предпочтения;

– **pedSink** – выход из модели.

Затем добавим некоторые элементы управления моделью, в которых будут задаваться параметры нашей модели (рисунок 16).

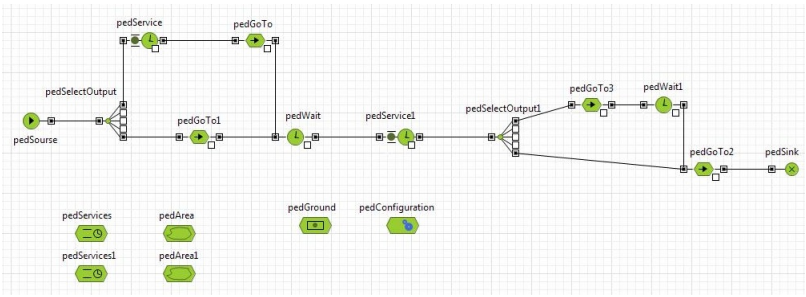

Рис. 16: Дополнительные объекты модели

Модель начинается с объекта **pedSourse**. Для него во вкладке свойств нам необходимо установить следующие параметры (рисунок 17): интенсивность пешеходов – 1000 в час, поставить галочку напротив параметра «ограничить прибытия» и в соответствующей графе установить значение 500 (среднее количество пассажиров самолёта).

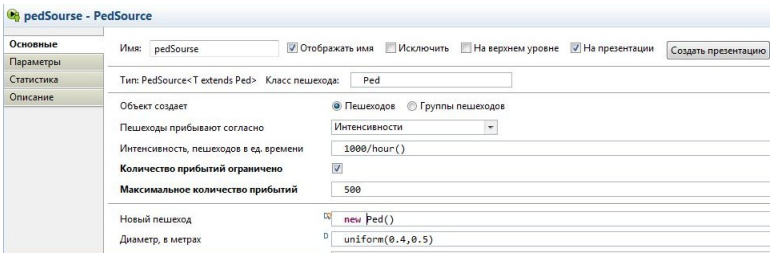

Рис. 17: Свойства объекта **pedSourse**

После того, как агент попал в модель (зашёл в аэропорт), происходит выбор инстанции, куда ему отправляться: с вероятностью 0,8 он пойдёт на стойку регистрации и с вероятностью 0,2 он отправится сразу в паспорт-

ный контроль (такой случай возможен, если пассажир купил и распечатал билет дома). Соответствующие данные записываем в свойства объекта **pedSelectOutput** (рисунок 18).

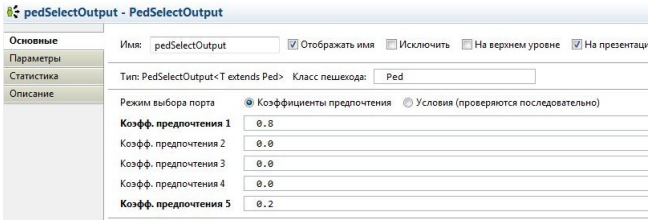

Рис. 18: Объект **pedSelectOutput**

Элемент нашей модели **pedService** отвечает за очередь к стойке регистрации, но параметры для него указываются в дополнительном объекте **pedServices**. Поэтому, в объекте **pedService** необходимо лишь указать сервис, в котором будут указаны все условия, которые должны выполняться на данном этапе в процессе работы модели (рисунок 19). Аналогично, объект **pedService1** (очередь на досмотр пассажиров) будет ссылаться на дополнительный объект **pedServices1**.

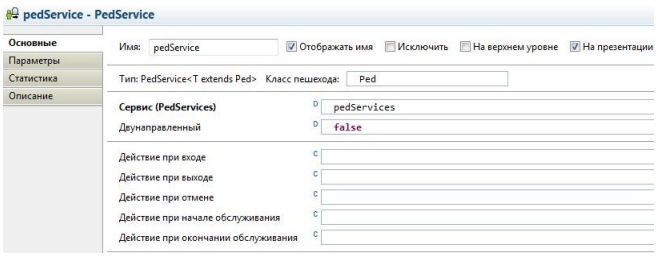

Рис. 19: Свойства элемента **pedService**

Дальше разберёмся с объектами pedGoTo. В нашем случае, их четыре и они задают следующие направления движения агентов:

– **pedGoTo** и **pedGoTo1** – направление к стойке паспортного контроля;

– **pedGoTo3** – к сдаче багажа;

– **pedGoTo2** – выход в зону вылета (выход из модели).

Для элементов **pedGoTo** и **pedGoTo1** свойства будут одинаковыми (рисунок 20). В случае **pedGoTo3** и **pedGoTo2** – цель, которая должна быть

достигнута агентом, будет отличаться (линии **line_baggage** и **output** соответственно). Остальные параметры оставляем без изменения.

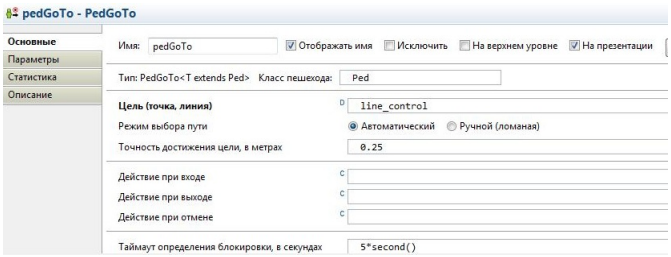

Рис. 20: Параметры объектов **pedGoTo** и **pedGoTo1**

Затем рассмотрим объект **pedWait** (паспортный контроль). Здесь необходимо указать время задержки агента у данной стойки. Так как проверка паспортных данных проходит достаточно быстро, установим время от 1 до 2 минут (рисунок 21), а область ожидания – **pedArea**. Отметим, что в модели указывается модельное время и одна секунда модельного времени равна одной минуте реального времени.

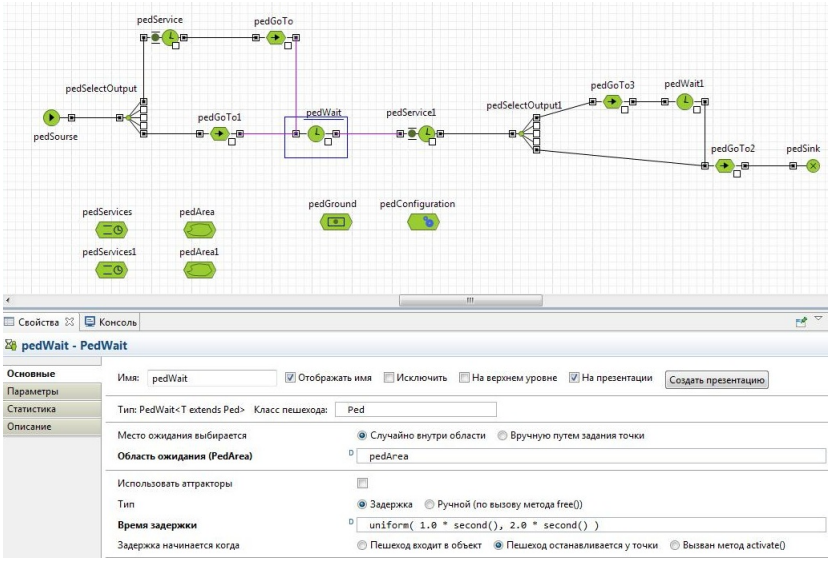

Рис. 21: Свойства объекта **pedWait**

В объекте **pedSelectOutput1** (рисунок 22) происходит выбор: агент направляется к пункту сдачи багажа (с вероятностью 0,9) или же у него отсутствует багаж (только ручная кладь, которую можно взять с собой в салон самолёта), с соответствующей вероятностью, равной 0,1.

Агенты, имеющие багаж, попадают в инстанцию сдачи багажа, соответствующую объекту модели **pedWait1**, где время ожидания варьируется от 0,5 до 1 минуты, а область ожидания – **pedArea1** (рисунок 23).

И последний элемент схемы нашей модели – **pedSink**. Здесь пока никакие изменения в свойства вносить не нужно. Рассмотрим теперь «дополнительные» объекты, необходимые для корректного выполнения нашей модели. В объектах типа **pedServices** нам необходимо указать следующие параметры: сервисы (стойки регистрации), время задержки (время обслуживания одного агента) и очереди. Для объекта **pedServices** данные параметры отображены на рисунке 24.

Аналогично заполняем свойства объекта **pedServices1** (рисунок 25).

В объекте **pedGround** нам необходимо указать стены нашего аэропорта (рисунок 26). За них в нашей модели отвечает группа линий **wall**.

Для объектов **pedArea** и **pedArea1** необходимо указать фигуру, в области которой будет проходить ожидание и этаж. В нашем случае, этаж для обоих элементов будет один и тот же – **pedGround**, а фигуры, внутри которых агенты будут ожидать обслуживания, **rectangle** (рисунок 27) и **rectangle1** соответственно.

Объект **pedConfiguration** оставляем без изменений.

Итак, все объекты нашей модели описаны. Теперь добавим функцию сбора статистики, с помощью которой мы определим среднее время прохода агентом всей модели (всех инстанций аэропорта). С палитры Статистика перетащим объект Статистика на рабочую область и дадим ей название **moveTime**. Для того чтобы данный элемент работал, нам необходимо добавить **Java**-класс. Для этого, во вкладке Проекты нажмём правой кнопкой мыши на название нашей модели, затем Создать → Java-класс. Дадим ему название **AeroPed**, а в качестве базового класса укажем класс **ped** (рисунок 28).

После этого нажимаем на кнопку Далее и в следующем окне (рисунок 29) добавим параметр **startMove**, в котором будет храниться время входа. Тип

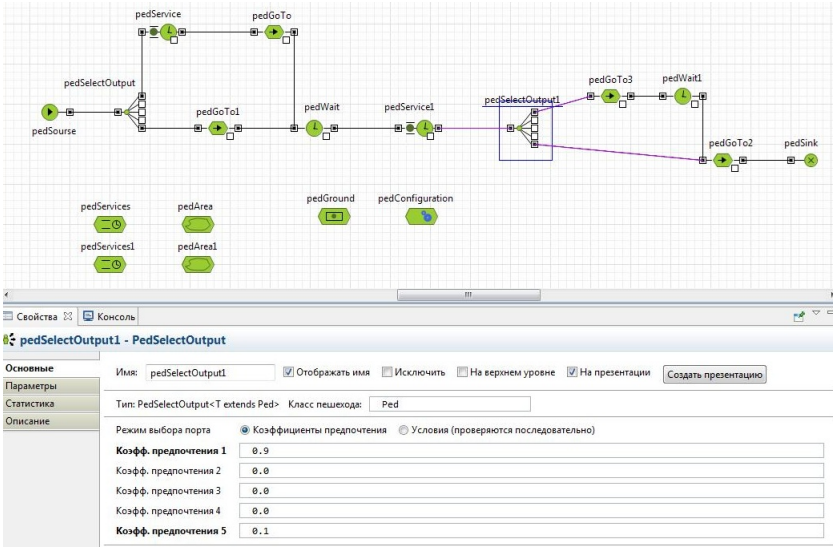

Рис. 22: Свойства элемента **pedSelectOutput1**

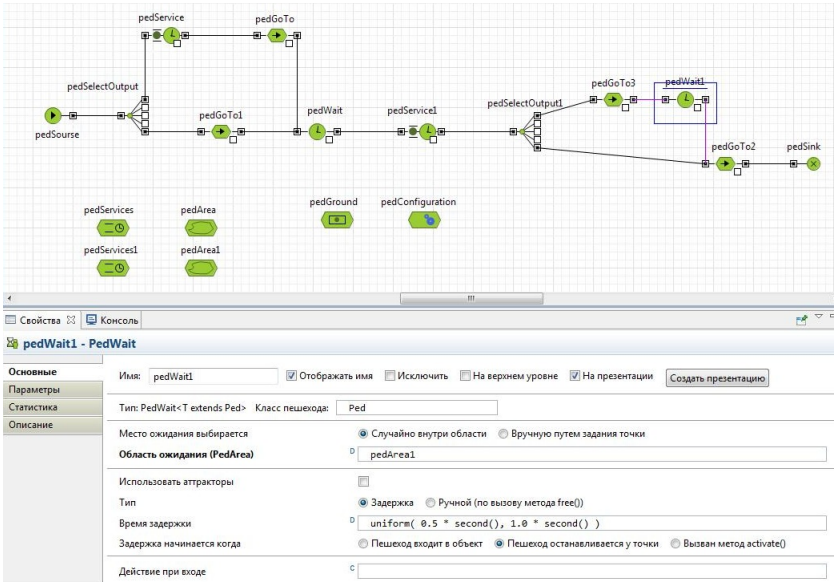

Рис. 23: Свойства объекта **pedWait1**

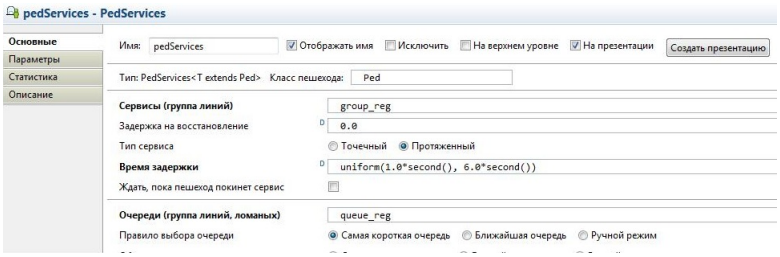

Рис. 24: Свойства объекта **pedServices**

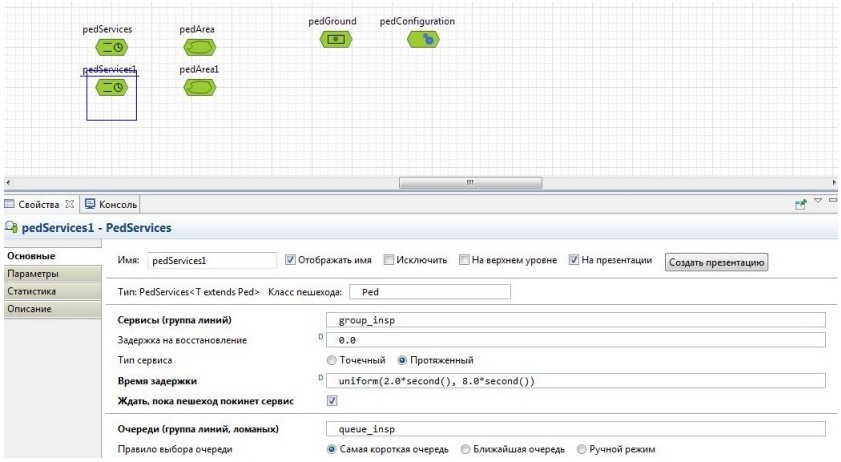

Рис. 25: Свойства объекта **pedServices1**

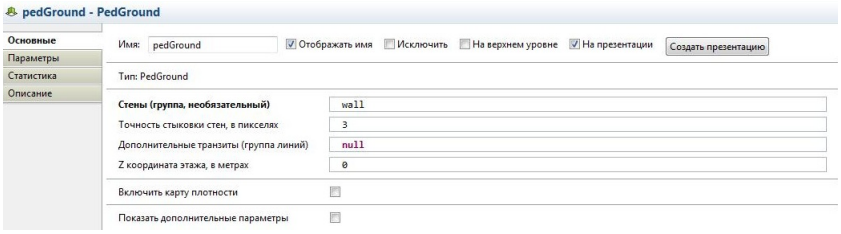

Рис. 26: Параметры объекта **pedGround**

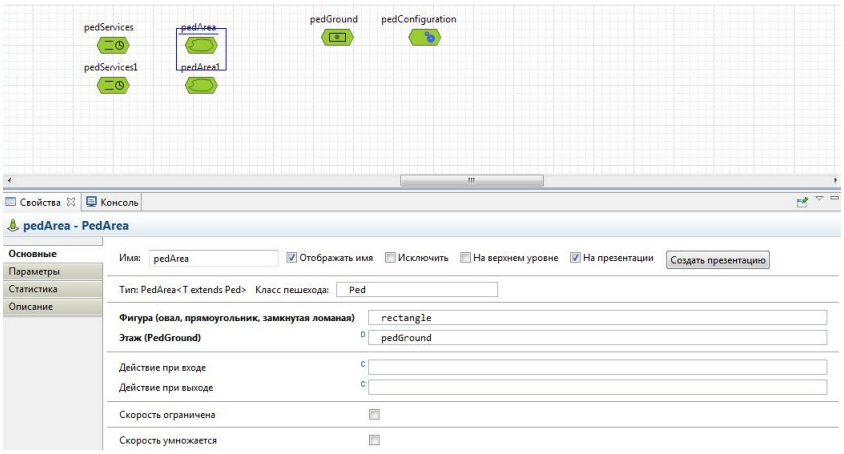

Рис. 27: Свойства объекта **pedArea**

данного параметра устанавливаем double, так как модельное время обрабатывается именно так.

Наш **Java**-класс готов. Теперь нам необходимо изменить параметры объекта источника:

- Класс пешехода заменим на **AeroPed**;
- В графе Новый пешеход **new Ped()** заменим на **new AeroPed()**;
- В графе Действие при выходе запишем время: **((AeroPed)ped).startMove = time()**;
- Для объекта **pedSink** необходимо сменить класс пешехода (также, как для объекта **pedSourse**) и укажем действие при входе, добавляющее данные к статистике: **moveTime.add(time()-((AeroPed)ped).startMove)**; данная функция высчитывает разницу текущего времени и времени входа агента в модель.

Запустим нашу модель и посмотрим результат её выполнения (рисунок 30).

Итак, в ходе прогона модели мы видим, что в пункте досмотра пассажиров у нас возникают огромные очереди, которые мешают пройти остальным агентам к их цели. При этом, среднее время прохождения агентом всех инстанций около 16 минут (рисунок 31).

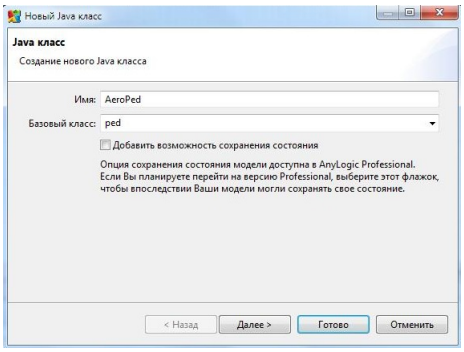

Рис. 28: Создание нового **Java**-класса

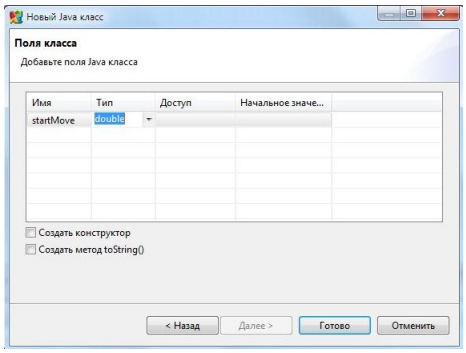

Рис. 29: Добавление параметра **startMove**

Рис. 30: Запуск модели

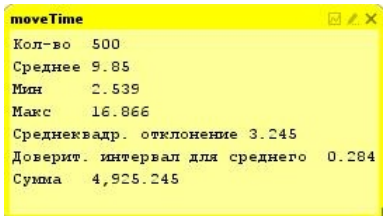

Рис. 31: Время обслуживания одного пассажира

Для исправления данной проблемы создадим ещё одну, четвертую очередь в пункте досмотра пассажиров. Для этого в палитре Презентация выберем инструмент Линия и достроим их, как показано на рисунках 32 и 33.

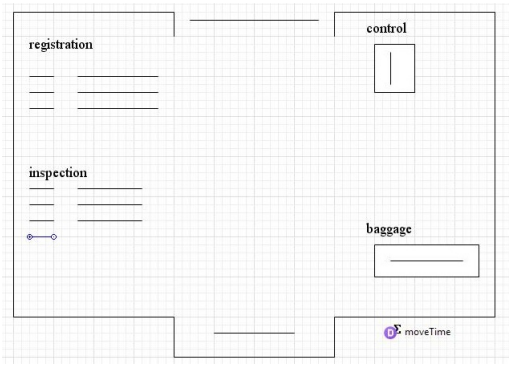

Рис. 32: Добавление стойки обслуживания в пункте досмотра пассажиров

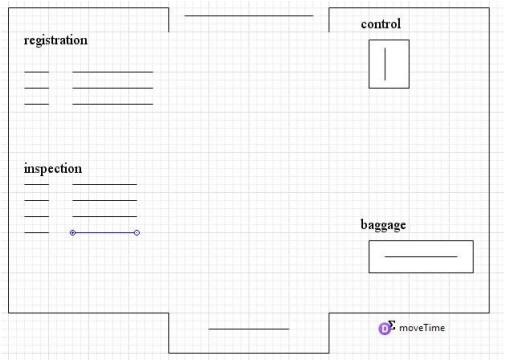

Рис. 33: Добавление очереди к стойке обслуживания в пункте досмотра пассажиров

Но для того, чтобы наши линии относились к нужным нам группам, необходимо сделать следующее: на линии, отвечающей за дополнительную стойку обслуживания, щелкнуть правой кнопкой мыши, выбрать Группировка → Добавить в существующую группу. В рабочей области появится следующее (рисунок 34):

Для добавления линии в группу достаточно щелкнуть на розовый кружок с делениями, относящийся к той группе, в которую мы хотим добавить линию (в нашем случае – это группа **group_insp**).

Аналогичные действия необходимо совершить для второй линии, но её нужно добавить в группу **queue_insp**. Попробуем снова запустить нашу модель (рисунок 35).

Очевидно, что проблема с огромными очередями в пункте досмотра решена. При этом, среднее время обслуживания одного агента снизилось до 8 минут (рисунок 36), что в два раза меньше, чем было изначально.

Исходя из данного факта, можно сделать вывод, что директору, для улучшения качества обслуживания пассажиров, необходимо добавить еще одну стойку в пункте досмотра пассажиров.

Дискретно-событийная модель работы полиграфа

Всем известно, что мы живём в век информационных технологий, когда всё, что нам интересно, можно найти в интернете. Тем не менее, бумажные издания остаются востребованными и ежегодно в свет только по одному направлению (естественно, пользующимся спросом) выходит около десяти тысяч новых книг. Процесс выхода книги в свет проходит по следующей схеме:

а) Рукопись передаётся в издательство, где она либо принимается, либо нет. При этом шанс отказа весьма мал (в силу огромного количества издательств, как больших и известных, так и наоборот);

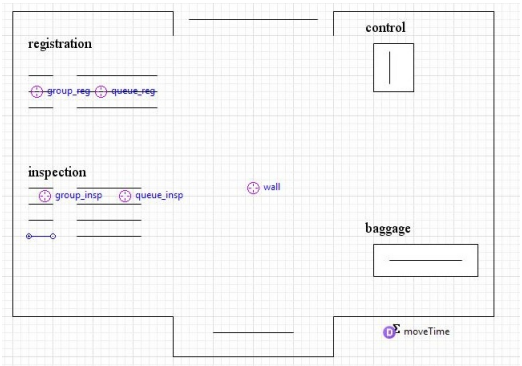

Рис. 34: Отображение всех групп, существующих в данной модели

б) Стадия допечатной подготовки (ввод и обработка текстовой и изобразительной информации, вёрстка и спуск полос, изготовление печатных форм);

в) Стадия печатной подготовки (подготовка печатной машины, печатание тиража, контроль качества печати, сдача отпечатанных листов в переплётно-брошюровочный цех);

г) Стадия послепечатной обработки (фальцовка, биговка, листоподборка, скрепление, упаковка и погрузка).

Но, несмотря на максимальную автоматизированность данного процесса, зачастую отдельные экземпляры, а то и весь тираж, попадают в брак (как по вине человека, отвечающего за тот или иной этап, так и по причине сбоя в технике).

Нам необходимо выяснить, сколько в среднем книг, с тиража объёмом в три тысячи экземпляров, попадают в брак. При этом мы учитываем, что максимально допустимый процент брака для заданного тиража - 7%.

Для этого мы построим дискретно-событийную модель.

Создадим чистую модель (Файл → Создать → Модель), на втором этапе выберем «с нуля» (рисунки 37 и 38).

Для дискретно-событийной модели воспользуемся объектами с закладки Основная библиотека (рисунок 39).

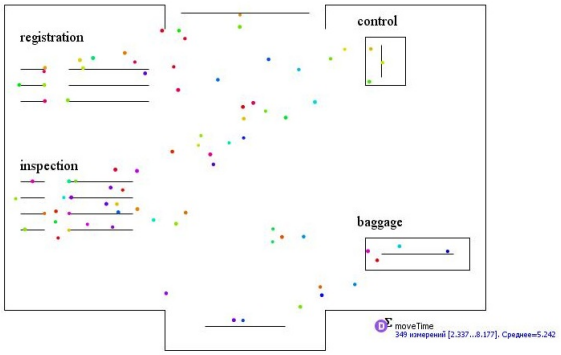

Рис. 35: Повторный запуск модели

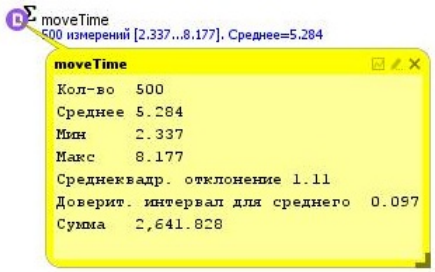

Рис. 36: Время обслуживания агентов

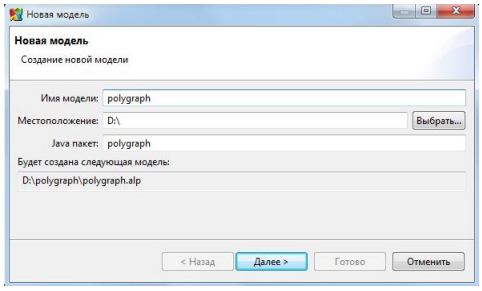

Рис. 37: Создание дискретно-событийной модели

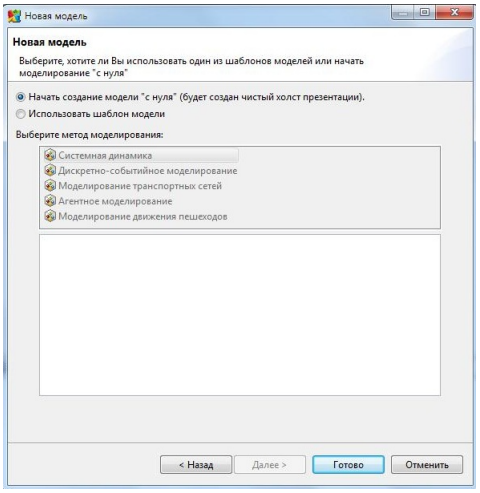

Рис. 38: Выбор пустого шаблона создания модели

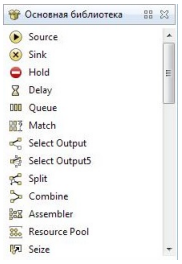

Рис. 39: Основная библиотека

Построение модели начинается с создания заявки: для этого из основной библиотеки перенесем объект **Sourse**. В нашей модели на основных этапах мы будем использовать объекты **Select Output** (обозначающий процесс, производимый на данном шаге) и Sink (Выход), а также объект **Split**.

Разместим объекты в рабочей области так, как показано на рисунке 40.

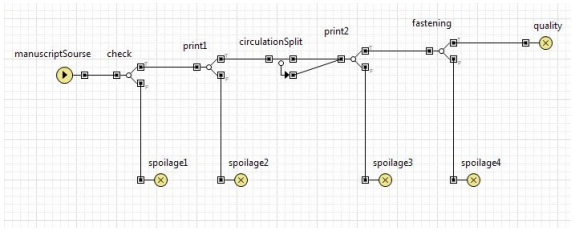

Рис. 40: Размещение объектов в рабочей области

Далее укажем параметры для каждого объекта. Для объекта **manuscriptSourse** ограничим количество прибытий (рисунок 41).

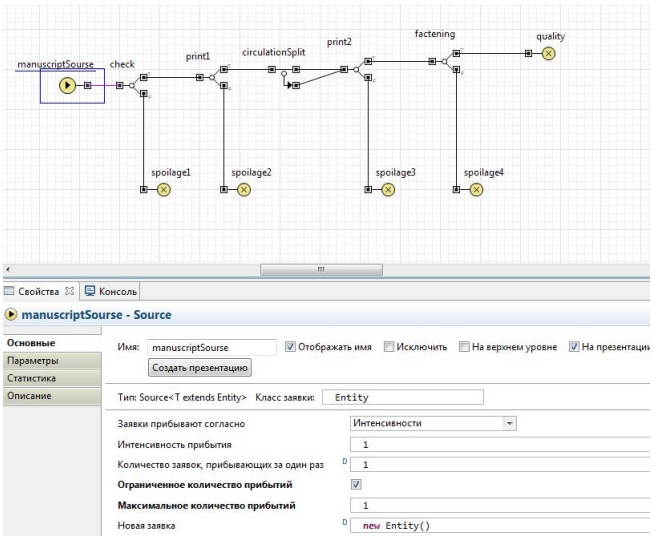

Рис. 41: Параметры объекта **Sourse**

Нам необходима одна заявка, попадающая на вход. В объекте **check** (**Select Output**) принимается решение: отправляется ли заявка (рукопись) в печать или же заявка признается негодной. Во втором случае происходит выход из модели, причем вероятность такого исхода очень мала. Поэтому в свойствах данного объекта в графе Вероятность мы указываем значение 0,99

(с такой вероятностью наша заявка пройдёт дальше, и лишь с вероятностью 0,01 заявка покинет модель). Остальные параметры оставляем без изменений.

Затем мы переходим к печати рукописи в количестве трёх тысяч экземпляров. Вообще говоря, если рукопись проходит проверку, то дальше она переходит на стадию допечатной подготовки, но для нашей модели этот шаг неважен, поэтому мы его опускаем.

Печать тиража в нашей модели разбита на три объекта:

а) Объект **print1**. Здесь мы проверяем (на глобальном уровне), прошёл ли тираж проверку качества: тираж напечатан качественно или тираж полностью признан бракованным. Выставляем для него параметры, как на рисунке 42

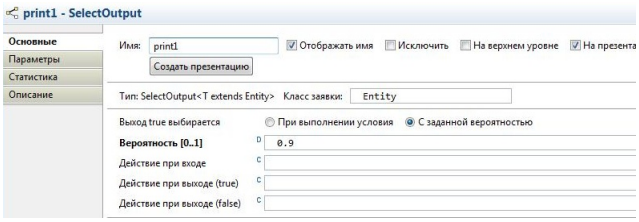

Рис. 42: Параметры объекта **print1**

б) Объект **circulationSplit**. Здесь мы "множим"нашу заявку, для того чтобы далее можно было проверять отдельно каждый экземпляр, в отличие от предыдущего шага, где мы смотрели на тираж как на единое целое. Так как мы рассматриваем тираж в 3000 экземпляров, а у нас есть одна заявка, нам необходимо сделать 2999 копий (рисунок 43).

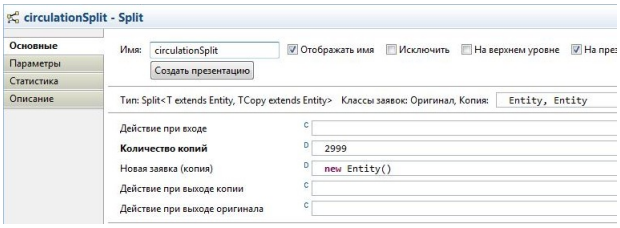

Рис. 43: Параметры объекта **Split**

в) Объект **print2**. На данном объекте мы проверяем отдельно каждую копию на качество. В случае брака данный экземпляр выходит из модели.

Здесь для задания вероятности мы используем функцию **uniform()** (рисунок 44).

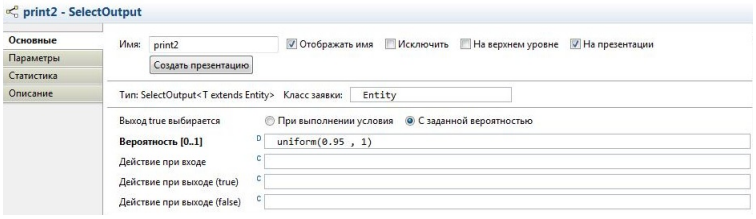

Рис. 44: Объект **print2**

Затем, в объекте **fastening**, начинается стадия послепечатной подготовки. Здесь шанс, что напечатанный экземпляр будет испорчен, невелик (рисунок 45). В случае, если экземпляр испорчен, он попадает на выход, в объект **spoilage4**.

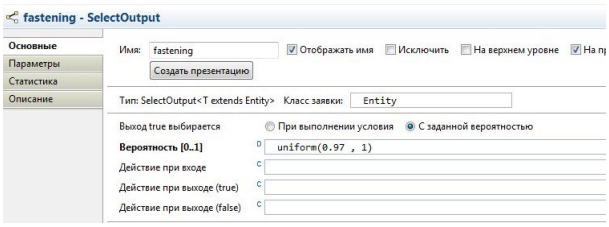

Рис. 45: Свойства объекта **fastening**

И, наконец, если копия прошла все стадии, не попав в брак, она выходит из модели «естественным» способом. В свойствах объекта **quality** зададим Действие при входе (рисунок 46). Этот параметр мы будем использовать в дальнейшем при построении диаграммы.

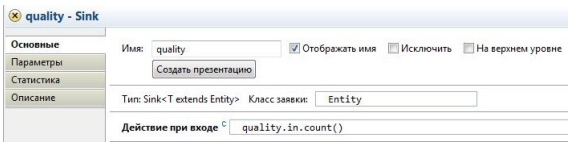

Рис. 46: Свойства объекта **quality**

Для сбора статистики добавим в нашу модель диаграмму, показывающую процент брака для данного тиража.

Для этого из палитры Статистика добавим объект **Круговая диаграмма** и зададим для него параметры, которые отображены на рисунке 47. На ней будет отображаться процентное количество бракованного и качественного продукта, полученного на выходе.

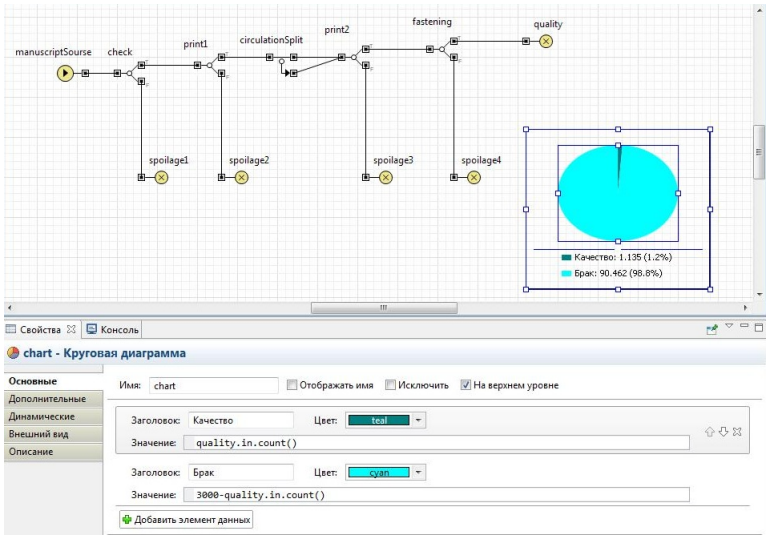

Рис. 47: Сбор данных

Запустим нашу модель (рисунок 48).

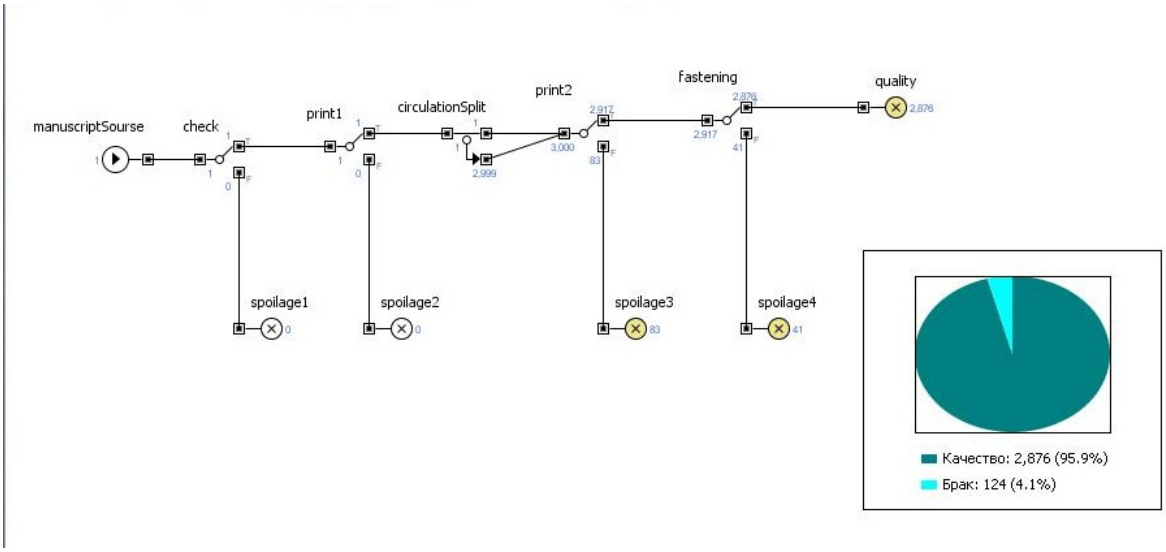

Рис. 48: Запуск модели

Таким образом, можно увидеть, что в подавляющем большинстве случаев количество брака не превышает допустимую норму.