Оглавление

Перечень сокращений

ИМ – Имитационное Моделирование

СДС – Сложная Дискретная Система

IDE - Integrated Development Environment (Интегрированная Среда Разработки)

СМО – Система Массового Обслуживания

Терминология

Плагин — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей.

Фреймворк — программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

1. Введение

Имитационное моделирование (ИМ)[1] на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами (СДС) и процессами, в них протекающими. К таким системам можно отнести экономические и производственные объекты, морские порты, аэропорты, комплексы перекачки нефти и газа, ирригационные системы, программное обеспечение сложных систем управления, вычислительные сети и многие другие. Широкое использование ИМ объясняется тем, что размерность решаемых задач и неформализуемость сложных систем не позволяют использовать строгие методы оптимизации. Эти классы задач определяются тем, что при их решении необходимо одновременно учитывать факторы неопределенности, динамическую взаимную обусловленность текущих решений и последующих событий, комплексную взаимозависимость между управляемыми переменными исследуемой системы, а часто и строго дискретную и четко определенную последовательность интервалов времени. Указанные особенности свойственны всем сложным системам.

Проведение имитационного эксперимента позволяет:

1. Сделать выводы о поведении СДС и ее особенностях:
    - без ее построения, если это проектируемая система;
    - без вмешательства в ее функционирование, если это действующая система, проведение экспериментов над которой или слишком дорого, или небезопасно;
    - без ее разрушения, если цель эксперимента состоит в определении пределов воздействия на систему.
2. Синтезировать и исследовать стратегии управления.
3. Прогнозировать и планировать функционирование системы в будущем.
4. Обучать и тренировать управленческий персонал и т.д.

ИМ является эффективным, но и не лишенным недостатков, методом. Трудности использования ИМ, связаны с обеспечением адекватности описания системы, интерпретацией результатов, обеспечением стохастической сходимости процесса моделирования, решением проблемы

размерности и т.п. К проблемам применения ИМ следует отнести также и большую трудоемкость данного метода.

Интеллектуальное ИМ, характеризующиеся возможностью использования методов искусственного интеллекта и прежде всего знаний, при принятии решений в процессе имитации, при управлении имитационным экспериментом, при реализации интерфейса пользователя, создании информационных банков ИМ, использовании нечетких данных, снимает часть проблем использования ИМ.

Разработка интеллектуальной среды имитационного моделирования РДО выполнена в Московском государственном техническом университете (МГТУ им.Н.Э. Баумана) на кафедре "Компьютерные системы автоматизации производства".

2. Предпроектное исследование

2.1.Основные положения языка РДО[3]

Основные положения языка РДО могут быть сформулированы следующим образом:

- Все элементы СДС представлены как ресурсы, описываемые некоторыми параметрами. Ресурсы могут быть разбиты на несколько типов; каждый ресурс определенного типа описывается одними и теми же параметрами.

- Состояние ресурса определяется вектором значений всех его параметров; состояние СДС - значением всех параметров всех ресурсов.

- Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным образом состояние ресурсов; действия ограничены во времени двумя событиями: событиями начала и событиями конца.

- Нерегулярные события описывают изменения состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отоношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.

- Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения состояния ресурсов в начале и в конце соответствующего действия.

- Множество ресурсов R и множество операций O образуют модель СДС.

2.2.Система имитационного моделирования RAO-XT

Система имитационного моделирования RAO-XT представляет собой плагин для интегрированной среды разработки Eclipse, позволяющий вести разработку имитационных моделей на языке РДО. Система написана на языке Java[6] и состоит из трех основных компонентов:

- rdo – компонент, производящий преобразование кода на языке РДО в код на языке Java.

- rdo.lib – библиотека системы. Этот компонент реализует ядро системы имитационного моделирования.
- rdo.ui – компонент, реализующий графический интерфейс системы с помощью библиотеки SWT[4].

2.3. **Синтаксический анализ[2]**

Анализ исходной программы разбивает её на составные части и накладывает на них грамматическую структуру. Анализ исходный программы разбивается на две фазы: лексический и синтаксический анализ. Лексический анализатор читает поток символов, составляющих исходную программу, и группирует эти символы в значащие последовательности, называемые лексемами. Для каждой лексемы анализатор строит выходной токен, состоящий из имени токена и значения его атрибута. Этот токен передается на следующую фазу – синтаксический анализ, которая также называется разбором (parsing). Синтаксический анализатор использует первые компоненты токенов, полученные при лексическом анализе для создания древовидного промежуточного представления, которое описывает грамматическую структуру потока токенов.

Синтаксис языка программирования описывает корректный вид его программы, а семантика определяет смысл написанной на нем программы. Иерархическая структура множества конструкций языка программирования, т.е. его синтаксис естественным образом описывается грамматикой. Синтаксис большинства языков программирования описывается контекстно-свободной грамматикой или BNF (Backus-Naur Form – Форма Бэкуса-Наура). Контекстно-свободная грамматика имеет 4 компонента:

- множество терминальных символов - элементарных символов языка, определяемых грамматикой, т.е. токенов;
- множество нетерминальных символов – синтаксических переменных;
- множество продукций, каждая из которых состоит из нетерминала, называемого заголовком или левой частью продукции, стрелки и последовательности терминалов или нетерминалов, называемых телом или правой частью продукции. Продукции определяют один из возможных видов конструкции языка;

- стартовый (начальный) символ – специально указанный нетерминальный символ.

Грамматика выводит (порождает) строки, начиная со стартового символа и неоднократно замещая нетерминалы телом продукции этих нетерминалов. Строки токенов, порождаемые из стартового символа, образуют язык, определяемый грамматикой.

Таким образом, синтаксический анализ представляет собой выяснение для полученной строки терминалов способа её вывода из стартового символа грамматики.

Для описания грамматики языка РДО используется фреймворк Xtext[5]. Xtext предоставляет язык описания грамматик, а также синтаксический анализатор, способный работать с контекстно-свободными грамматиками.

3. Формирование ТЗ

3.1.Введение

Программный комплекс RAO-XT предназначен для разработки и отладки имитационных моделей на языке РДО. Основные цели данного комплекса - обеспечение пользователя легким в обращении, но достаточно мощным средством разработки текстов моделей на языке РДО, обладающим большинством функций по работе с текстами программ, характерных для сред программирования, а также средствами проведения и обработки результатов имитационных экспериментов.

3.2. Общие сведения

Основание для разработки: задание на курсовой проект.

Заказчик: Кафедра «Компьютерные системы автоматизации производства» МГТУ им. Н.Э. Баумана

Разработчик: студент кафедры «Компьютерные системы автоматизации производства» Богачев П.А.

Наименование темы разработки: «Новая грамматика языка РДО»

3.3. Назначение разработки

Разработать новую грамматику для языка имитационного моделирования РДО, интегрировать данную грамматику в среду разработки RAO-XT и написать документацию по новой грамматике.

3.4. Требования к программе или программному изделию

3.4.1. Требования к функциональным характеристикам

Новая грамматика должна решить следующие проблемы старой грамматики:

- Несоответствие стилей разных компонентов языка.
- Перегруженность ключевыми словами.
- Слишком большой размер получающихся моделей.
- Несоответствие синтаксиса некоторых конструкций их смысловому содержанию.
- Большое количество искусственных ограничений.

3.4.2. Требования к надежности

Основное требование к надежности направлено на поддержание в исправном и работоспособном состоянии ЭВМ, на которой происходит использование программного комплекса RAO-XT.

3.4.3. Условия эксплуатации

- Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости.
- Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением.

3.4.4. Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 2 Гб;
- объем жесткого диска не менее 50 Гб;
- микропроцессор с тактовой частотой не менее 2ГГц;
- монитор с разрешением от 1368 * 768 и выше.

3.4.5. Требования к информационной и программной совместимости

Система должна работать под управлением следующих ОС: Windows 7, Ubuntu 14.10.

3.4.6. Требования к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются.

3.4.7. Требования к транспортированию и хранению

Требования к транспортированию и хранению не предъявляются.

3.5. Требования к программной документации

Документация по новой грамматике должна состоять из двух компонентов: руководство по изучению языка и формальное описание синтаксиса языка.

3.6.Стадии и этапы разработки

Плановый срок начала разработки – 7 февраля 2015г.

Плановый срок окончания разработки – 25 мая 2015г.

Этапы разработки:

- Концептуальный этап проектирования системы;
- Технический этап проектирования системы;
- Рабочий этап проектирования системы.

3.7.Порядок контроля и приемки

Контроль и приемка работоспособности системы осуществляются с помощью ручного тестирования. Создаются модели в новой грамматике и результаты моделирования сравниваются с результатами аналогичных моделей в старой грамматике.

4. Концептуальный этап проектирования системы

На концептуальном этапе необходимо разработать новую грамматику языка. Стилистически новая грамматика должна соответствовать языкам Java и Xtend, на которых основана система.

Грамматика была обновлена для следующих компонентов языка РДО:

- Объявления типов ресурсов;
- Создания ресурсов;
- Описания образцов;
- Описания событий;
- Описания точек принятия решений;
- Описания последовательностей;
- Описания функций;
- Описания констант;
- Описания результатов;
- Инициализации прогона;

4.1.Грамматика объявления типов ресурсов

На момент начала работы грамматика объявления типов ресурсов имела следующий вид:

```
$Resource_type Клиенты : temporary
$Parameters
        тип      : ( Тип1, Тип2 )
        состояние: ( Пришел, Начал_стрижку )
$End

$Resource_type Парикмахеры: permanent
$Parameters
        состояние_парикмахера : ( Свободен, Занят ) = Свободен
        количество_обслуженных: integer
        длительность_min      : integer
        длительность_max      : integer
        тип_клиента           : such_as Клиенты.тип
$End
```

Данная грамматика обладает следующими недостатками:

- Объявление перегружено неудобными ключевыми словами. Было принято решение использовать ключевое слово type вместо *$Resource_type*, а список параметров заключать в фигурные скобки, в результате чего ключевые слова *$Parameters* и *$End* могут быть удалены.

- Объявление типа ресурса постоянным или временным неудобно, так как разные ресурсы одного и того же типа могут фигурировать в модели как постоянно, так и на коротком промежутке модельного времени. Наиболее удобным решение представляется объявление всех типов ресурсов как временых и снятие связанных с этим ограничений.
- Типы данных параметров типа ресурса не соотносятся с типами данных языка Java, на основе которого построена система RAO-XT. Следует изменить имена типов данных на следующие:
  - *integer -> int*
  - *real -> double*
  - *string -> String*
- Стиль объявление параметров, при которым тип следует после имени сейчас является устаревшим. Кроме того, данный стиль не согласуется со стилем языка Java, в котором имя параметра следует после типа параметра. Следует перейти на стиль языка Java. Параметры должны разделяться точками с запятой.
- Объявление перечислимого типа производится непосредственно при объявлении параметра, при этом перечислимые типы не имеют собственных имен. В результате для объявление двух параметров, имеющих одинаковый перечислимый тип, необходимо из одного параметра ссылаться на другой с помощью конструкции *such_as*. Эту схему следует изменить следующим образом:
  - объявлять перечислимые типы отдельно от параметров ресурса
  - назначать перечислимым типам имена
  - ссылаться на имя перечислимого типа, а не на имя параметра ресурса
  - удалить конструкцию *such_as*
- Отсутствует возможность создавать ресурс без имени. В новой грамматике для этого следует вызывать метод *create()* у типа ресурса. но не связывать полученный результат ни с каким именем.

В результате объявление типа ресурса из примера выше будет иметь следующий вид:

```
enum Тип_клиента {Тип1, Тип2}
enum Состояние_клиента {Пришел, Начал_стрижку}
enum Состояние_парикмахера {Свободен, Занят}

type Клиенты {
        Тип_клиента тип;
        Состояние_клиента состояние;
}
```

```
type Парикмахеры {
        Состояние_парикмахера                    состояние_парикмахера                    =
Состояние_парикмахера.Свободен;
        int количество_обслуженных;
        int длительность_min;
        int длительность_max;
        Тип_клиента тип_клиента;
}
```

4.2.Грамматика создания ресурсов

На момент начала работы грамматика создания ресурсов имела следующий вид:

```
$Resources
        Парикмахерская = Парикмахерские(0);
        Парикмахер_1 = Парикмахеры(*, 0, 20, 40, Тип1);
        Парикмахер_2 = Парикмахеры(*, 0, 25, 70, Тип2);
        Парикмахер_3 = Парикмахеры(*, 0, 30, 60, Тип2);
$End
```

Данная грамматика уже была частично переработана ранее, однако все еще имеет некоторые недостатки:

- Создание ресурсов таким образом может производиться только до начала прогона. Конструкия создания ресурсов в событиях и образцах координально отличается (см. 4.3, 4.4). Следует унифицировать конструкцию создания ресурсов для любого места модели. Стиль объявления ресурсов должен соответствовать стилю языка выражений, осованному на языка Java. Для этого было принято решение использовать ключевое слово create, обозначающее метод у типа ресурса. Тип ресурса в данной конструкции играет роль фабрики. Объединять создание ресурсов в группы не нужно.
- Для избежания коллизий имен перечислимых типов следует использовать полные имена для задания значений параметров этих типов.

В результате описание создания ресурсов из примера выше будет иметь следующий вид:

```
resource Парикмахерская = Парикмахерские.create(0);
resource Парикмахер_1 = Парикмахеры.create(*, 0, 20, 40, Тип_клиента.Тип1);
resource Парикмахер_2 = Парикмахеры.create(*, 0, 25, 70, Тип_клиента.Тип2);
resource Парикмахер_3 = Парикмахеры.create(*, 0, 30, 60, Тип_клиента.Тип2);
```

4.3.Грамматика описания образцов

На момент начала работы грамматика описания образцов имела следующий вид:

```
$Pattern Образец_обслуживания_клиента : operation
$Relevant_resources
        _Парикмахерская: Парикмахерская Keep NoChange
        _Клиент       : Клиенты       Keep Erase
        _Парикмахер   : Парикмахеры   Keep Keep
$Time = Длительность_обслуживания( _Парикмахер.длительность_min,
                _Парикмахер.длительность_max )
$Body
_Парикмахерская:
        Choice from _Парикмахерская.количество_в_очереди > 0
        Convert_begin
                количество_в_очереди--;

_Клиент:
        Choice from _Клиент.состояние == Пришел
        Convert_begin
                состояние = Начал_стрижку;

_Парикмахер:
        Choice from _Парикмахер.состояние_парикмахера == Свободен and
                _Парикмахер.тип_клиента == _Клиент.тип
        with_min( _Парикмахер.количество_обслуженных )
        Convert_begin
                состояние_парикмахера  = Занят;
        Convert_end
                состояние_парикмахера  = Свободен;
                количество_обслуженных++;
$End
```

Данная грамматика обладает следующими недостатками:

- Конструкция подбора релевантных ресурсов отделена от конструкции их объявления. Следует описать правила подбора в том же месте, где релевантным ресурсам задаются имена. Конструкцию подбора релеватных ресурсов следует представить в виде вызова метода *select()* у типа ресурса, а также методов *withMin()*, *withMax()* или *first()* над списков ресурсов, которые могут быть подобраны как релевантные, который был возвращен методов *select()*.
- Для каждого релевантного ресурса задается отдельное тело (конвертер) внутри образца с описаниями действий над этим релевантным ресурсов в начале и в конце операции или при выполении правила. Следует определить для операции методы *begin()* и *end()*, а для продукционных правил метод *execute()*, в теле которых будут описываться действия над всеми релевантными ресурсами образца сразу. Методы будут переопределяться для каждого образца с помощью ключевого слова set.

- Задание длительности операции следует так же вынести в отдельный метод *duration()*, аналогичный методам, описанным выше. При этом порядок переопределения методов внутри образца неважен, однако методы не могут быть переопределены более одного раза.
- В результате проведения описанных выше изменений ограничения, задаваемые статусами конверторов (*Keep*, *NoChange*, и т.д.) теряют смысл, и потому могут быть удалены. Конверторы *Create* и *Erase*, используемые для создания и удаления ресурсов в образцах должны быть заменены конструкциями внутри переопределенных методов образцов. Конструкция создания ресурса была описана в пункте 4.2, а конструкция удаления ресурса должна иметь аналогичный ей формат. Для этого следует использовать вызов метода *erase()*.
- Механизм релевантных ресурсов используется даже для обращения к параметрам ресурсов, которые не надо подбирать. Следует разрешить обращение к таким параметрам по глобальному имени ресурса без объявления его как релевантного.
- Ключевые слова образцов, как и для других компонентов языка, следует упростить и свести к единому стилю.
- Так как образец может иметь параметров, то в случае их отсутствия следует так же ставить скобки после имени образца, но пустые.

В результате описание операции из примера выше будет иметь следующий вид:

```
operation Образец_обслуживания_клиента() {
        relevant _Парикмахерская = Парикмахерская.select(_Парикмахерская.количество_в_очереди > 0);
        relevant _Клиент = Клиенты.select(_Клиент.состояние == Состояние_клиента.Пришел);
        relevant _Парикмахер = Парикмахеры.select(_Парикмахер.состояние_парикмахера ==
                Состояние_парикмахера.Свободен and _Парикмахер.тип_клиента ==
                _Клиент.тип).withMin(_Парикмахер.количество_обслуженных);

        set duration() {
                return Длительность_обслуживания.next(
                        _Парикмахер.длительность_min, _Парикмахер.длительность_max);
        }

        set begin() {
                _Парикмахерская.количество_в_очереди--;
                _Клиент.состояние = Состояние_клиента.Начал_стрижку;
                _Парикмахер.состояние_парикмахера = Состояние_парикмахера.Занят;
        }

        set end() {
```

```
            _Парикмахер.состояние_парикмахера = Состояние_парикмахера.Свободен;
            _Парикмахер.количество_обслуженных++;
            _Клиент.erase();
        }
}
```

4.4.Грамматика описания событий

На момент начала работы грамматика описания событий имела следующий вид:

```
$Pattern Образец_прихода_клиента : event
$Relevant_resources
        _Парикмахерская: Парикмахерская Keep
        _Клиент       : Клиенты       Create
$Body
_Парикмахерская:
        Convert_event
            Образец_прихода_клиента.planning( time_now + Интервал_прихода( 30 ) );
            количество_в_очереди++;

_Клиент:
        Convert_event
            тип       = Тип_клиента;
            состояние = Пришел;
$End
```

В текущей грамматике языка РДО события являются особым типом образцов. Однако концептуально это не так: события – это элементы событийного подхода к имитационному моделированию, а образцы – элементы подхода сканирования активностей. Их мнимая похожесть в языке РДО объяснялась необходимостью использования механизма релевантных ресурсов для создания ресурсов в процессе прогона или обращения к параметрам ресурсов, созданных до начала прогона и имеющих имена. В результате изменений проведенных в предыдущих пунктах модели, появилась возможность полностью разделить события и образцы.

В результате описание события из примера выше будет иметь следующий вид:

```
event Событие_прихода_клиента() {
        Клиенты.create(Случайный_тип_клиента.next(), Состояние_клиента.Пришел);
        Событие_прихода_клиента.plan(currentTime + Интервал_прихода.next());
        Парикмахерская.количество_в_очереди++;
}
```

4.5.Грамматика описания точек принятия решений

На момент начала работы грамматика описания точек принятия решений типа some имела следующий вид:

```
$Decision_point model: some
$Activities
        Обслуживание_клиента: Образец_обслуживания_клиента;
$End
```

Грамматика описания точек принятия решений типа search имела следующий вид:

```
$Decision_point Расстановка_фишек : search
$Condition Exist(Фишка: Фишка.Номер <> Фишка.Местоположение)
$Term_condition
        For_All(Фишка: Фишка.Номер == Фишка.Местоположение)
$Evaluate_by 0
$Compare_tops = YES
$Activities
        Перемещение_вправо: Перемещение_фишки(справа,  1) value after 1;
        Перемещение_влево : Перемещение_фишки (слева, -1) value after 1;
        Перемещение_вверх : Перемещение_фишки (сверху,-3) value after 1;
        Перемещение_вниз  : Перемещение_фишки (снизу,  3) value after 1;
$End
```

Помимо этого имелись так же точки принятия решений типа *prior*, во всем аналогичные точкам типа *some*, но обладающие дополнительными возможностями задания приоритета.

Данная грамматика обладает следующими недостатками:

- Точки принятия решений *some* и *prior* описываются различными конструкциями. При этом точки принятия решений типа some являются частным случае точек принятия решения типа *prior* (в случае одинаковых приоритетов всех активностей). Следует использовать для них единую конструкцию с опциональным заданием приоритета. Если приоритет не задан, точки принятия решений будут работать так же, как some.
- Точки принятия решений типа *search* имеют большое количество характеристик, которые необходимо сконфигурировать при инициализации точек. При этом задание этих характеристик производится в строгом но не имеющим определенного логического обоснования порядке. Следует организовать задание параметров точки в виде вызова соответствующих методов в произвольном порядке. Сами вызовы будут производиться в теле метода *init()* точки принятия решений.

- Грамматику списка активностей необходимо привести к единому стилю с уже преобразованной грамматикой. Стиль конструкций *value before* и *value after* следует свести к стилю вызову методов.

В результате описание точек принятия из примера выше будет иметь следующий вид:

```
dpt model {
        activity Обслуживание_клиента checks Образец_обслуживания_клиента();
}

search Расстановка_фишек {
        set init() {
                setCondition(exist(Фишка: Фишка.Номер != Фишка.Местоположение));
                setTerminateCondition(forAll(Фишка: Фишка.Номер == Фишка.Местоположение));
                compareTops(true);
                evaluateBy(0);
        }
        activity Перемещение_вправо checks Перемещение_фишки(
                Место_дырки.справа,  1).setValueAfter(1);
        activity Перемещение_влево checks Перемещение_фишки(
                Место_дырки.слева,  -1).setValueAfter(1);
        activity Перемещение_вверх checks Перемещение_фишки(
                Место_дырки.сверху,  -3).setValueAfter(1);
        activity Перемещение_вниз checks Перемещение_фишки(
                Место_дырки.снизу,  3).setValueAfter(1);
}
```

4.6.Грамматика описания последовательностей

На момент начала работы грамматика описания точек принятия решений типа some имела следующий вид:

```
$Sequence Интервал_прихода : real
$Type = exponential 123456789
$End

$Sequence Длительность_обслуживания : real
$Type = uniform 123456789
$End
```

Данная грамматика обладает следующими недостатками:

- Излишние ключевые слова.
- Невозможность задания параметров распределения на этапе объявления последовательности. Следует обеспечить возможность задать параметры как в момент объявления последовательности, так и в момент вызова метода получения следующего значения

последовательности. Получение следующего значения последовательности следует реализовать через вызов метода next().

- Для экспоненциального распределения в качестве параметра задается среднее арифметическое, а не интенсивность. Задание распределения через интенсивность является более общепринятым вариантом, и реализовать следует именно его.

В результате описание последовательности из примера выше будет иметь следующий вид:

```
sequence Интервал_прихода = double exponential(123456789, 1/30.0);
sequence Длительность_обслуживания = double uniform(123456789);
```

4.7.Грамматика описания функций

На момент начала работы грамматика описания функций имела следующий вид:

```
$Function Фишка_на_месте : integer
$Type = algorithmic
$Parameters
        _Номер: such_as Фишка.Номер
        _Место: such_as Фишка.Местоположение
$Body
        if (_Номер == _Место)
                return 1;
        else
                return 0;
$End
```

Внутри тела функция использует Java-подобный язык выражений, однако сама структура объявления функции имеет совершенно другой стиль. Этот стиль перегружен ключевыми словами и в целом неудобен. Следует свести стиль объявления функций к Java-стилю.

В результате описание функции из примера выше будет иметь следующий вид:

```
int Фишка_на_месте(int _Номер, int _Место) {
        if (_Номер == _Место)
                return 1;
        else
                return 0;
}
```

4.8.Грамматика описания констант

На момент начала работы грамматика объявления констант имела следующий вид:

```
$Constant
        Длина_поля : integer = 3
$End
```

Грамматику объявления констант следует свести к стилю остальных компонентов языка, уже описанных выше. В результате объявление констант из примера выше будет иметь следующий вид:

```
constant int Длина_поля = 3;
```

4.9.Грамматика инициализации модели

На момент начала работы грамматика объявления результатов имела следующий вид:

```
$Results
        Занятость_парикмахера_1 : watch_state Парикмахер_1.состояние_парикмахера ==
Занят
        Занятость_парикмахера_2 : watch_state Парикмахер_2.состояние_парикмахера ==
Занят
        Занятость_парикмахера_3 : watch_state Парикмахер_3.состояние_парикмахера ==
Занят
        Обслужено_парикмахером_1: get_value Парикмахер_1.количество_обслуженных
        Обслужено_парикмахером_2: get_value Парикмахер_2.количество_обслуженных
        Обслужено_парикмахером_3: get_value Парикмахер_3.количество_обслуженных
$End
```

Грамматику объявления результатов следует свести к стилю остальных компонентов языка, уже описанных выше. В результате объявление результатов из примера выше будет иметь следующий вид:

```
result Занятость_парикмахера_1 = watchState(Парикмахер_1.состояние_парикмахера ==
                Состояние_парикмахера.Занят);
result Занятость_парикмахера_2 = watchState(Парикмахер_2.состояние_парикмахера ==
                Состояние_парикмахера.Занят);
result Занятость_парикмахера_3 = watchState(Парикмахер_3.состояние_парикмахера ==
                Состояние_парикмахера.Занят);
result Обслужено_парикмахером_1 = getValue(Парикмахер_1.количество_обслуженных);
result Обслужено_парикмахером_2 = getValue(Парикмахер_2.количество_обслуженных);
result Обслужено_парикмахером_3 = getValue(Парикмахер_3.количество_обслуженных);
```

4.10. Грамматика инициализации модели

На момент начала работы грамматика инициализации модели имела следующий вид:

```
$Simulation_run
```

```
        Образец_прихода_клиента.planning( time_now + Интервал_прихода( 30 ) );
        Terminate_if Time_now >= 360;
$End
```

В данной конструкции смешаны конструкции из языка выражений и специфические конструкции РДО. Необходимо сделать следующее:

- Разделить высказывания, связанные с инициализацией модели, и задание условия завершения модели.
- Высказывания, связанные с инициализацией модели определять в глобальном методе *init()*.
- Вычисление условия окончания моделирования осуществлять в глобальном методе *terminateCondition()*.

В результате описание инициализации модели из примера выше будет иметь следующий вид:

```
set init() {
        Событие_прихода_клиента.plan(currentTime + Интервал_прихода.next());
}

set terminateCondition() {
        return currentTime >= 360;
}
```

В Приложении 1 приведен полный код модели многоканальной СМО в старой грамматике. В Приложении 2 приведен полный код той же модели в новой грамматике.

5. Технический этап проектирования системы

На техническом этапе требовалось разработать грамматические правила на языке описания грамматик XText, которые позволили бы описать новую грамматику РДО.

5.1. Новые глобальные компоненты языка

Тело продукции стартового символа грамматики представляет собой множество возможных глобальных компонентов языка. Для описания новых компонентов языка к телу продукции стартового символа были добавлены следующие нетерминалы: *DefaultMethod* и *EnumDeclaration*. Помимо этого, события стали включаться в модель непосредственно, а не через грамматику образцов. В результате грамматические правила для описания глобальных компонентов модели выглядят следующим образом:

```
RDOModel:
      {RDOModel} objects += RDOEntity*
;

RDOEntity
      : ResourceType
      | Resources
      | Sequence
      | Constant
      | Function
      | Event
      | Pattern
      | DecisionPoint
      | Frame
      | Result
      | DefaultMethod
      | EnumDeclaration
;
```

Грамматические правила переопределения стандартных методов модели имеет следующий вид:

```
DefaultMethod:
      'set' name = ID '(' ')' '{'
            body = StatementList
      '}'
   ;
```

При этом тело метода описывается набором произвольных высказываний на языке выражений.

Грамматические правила для объявления перечислимых типов выглядят следующим образом:

```
EnumDeclaration:
	'enum' name = ID '{' values += EnumID (',' values += EnumID)* '}'
;
```

5.2.Грамматика ресурсов

Грамматические правила для объявления типов ресурсов выглядят следующим образом:

```
ResourceType:
	'type' name = ID '{'
		(parameters += ParameterType ';' )+
	'}'
;
```

Благодаря более обобщенному описанию объектов перечислимого типа, а также благодаря приведению синтаксиса описания параметров к Java-стилю, удалось упростить грамматику для описания параметров различных типов.

```
ParameterType
	: ParameterTypeBasic
	| ParameterTypeString
	| ParameterTypeArray
;

ParameterTypeBasic:
	type = (RDOInt | RDODouble | RDOBoolean | RDOEnum) name = ID
		('=' default = ExpressionOr)?
;

ParameterTypeString:
	type = RDOString name = ID ('=' default = STRING)?
;

ParameterTypeArray:
	type = RDOArray name = ID ('=' default = ArrayValues)?
;
```

Грамматические правила создания ресурсов сводятся к вызову единого высказывания создания ресурса *ResourceCreateStatement*. Высказывание позволяет как создавать ресурсы, имеющие имена, так и безымянные ресурсы. Любой ресурс, созданный таким образом, добавляется в базу данных модели.

```
Resources:
	resource = ResourceCreateStatement ';'
;

ResourceCreateStatement:
	('resource' name = ID '=')? reference = [ResourceType|FQN]
```

```
            '.' 'create' '(' ( parameters = ResourceExpressionList )?
')'
;

ResourceEraseStatement:
      relres = [RelevantResource | FQN] '.' 'erase' '('')'
;
```

Данное высказывание, как и высказыване удаления ресурса, было добавлено к общему списку возможных высказываний языка, и может быть использовано в любом месте, где возможно использование языка выражений, в частности в теле любого события или образца.

5.3.Грамматика описания событий

Описание событий было грамматически отдельно от описания остальных образцов. Из грамматики событий была удалена возможность объявления релевантных ресурсов, что позволило максимально упростить их синтаксис:

```
Event:
      'event' name = ID '('
            (parameters += ParameterType (
                  ',' parameters += ParameterType)* )? ')' '{'
            body = StatementList
      '}'
;
```

В новой грамматике события имеют только список параметров и тело, описываемое набором произвольного количества высказываний на языке выражений.

5.4.Грамматика описания образцов

Вынесение событий в отдельный компонент позволило свести все виды оставшихся образцов к единому синтаксису. В теле образцов переопределяется произвольное количество методов образца. Корректность переопределений валидируется синтаксическим анализатором в процессе написания модели (проверка корректности не требует компиляции модели).

```
Pattern:
      type = PatternType name = ID '('
            (parameters += ParameterType (
                  ',' parameters += ParameterType)* )? ')' '{'
            relevantresources += RelevantResource*
            ('relevantSet' combinational = PatternSelectMethod ';')?
            defaultMethods += DefaultMethod*
      '}'
;
```

```
enum PatternType
      : OPERATION = 'operation'
      | RULE = 'rule'
      | KEYBOARD = 'keyboard'
;
```

Из грамматики подбор релевантных ресурсов были удалены правила для задания статусов конверторов.

```
RelevantResource:
      'relevant' name = ID '=' type = [META_RelResType|FQN]
      (haveselect ?= '.' 'select' '(' select = PatternSelectLogic ')'
            (selectmethod = PatternSelectMethod)?
      )? ';'
;
```

5.5.Грамматика описания точек принятия решений

Точки принятия решений типа prior были грамматически сведены к точкам принятия решений типа some и объявляются в новой грамматике с помощью ключевого слова *dpt*. Точки принятия решений, реализующие поиск по графу, объявляются с помощью ключевого слова *search*. Грамматика, описывающая простые точки принятия решений приведена ниже:

```
DecisionPoint
      : DecisionPointSome
      | DecisionPointSearch
;

DecisionPointSome:
      'dpt' name = ID '{'
            ('set' initName = ID '(' ')' '{'
                  (initStatements += DptStatements ';')*
            '}')?
            ( activities += DecisionPointActivity ';')+
      '}'
;

DecisionPointActivity:
      'activity' name = ID ('(' priority = ExpressionOr ')')?
      'checks' pattern = [Pattern|FQN]
      '(' ( parameters += ExpressionOr (
                  ',' parameters += ExpressionOr)* )? ')'
;
```

Грамматика точек принятия решений типа *search* во многом аналогична.

Для точек принятия решений был написан ряд возможных инициализирующих высказываний, которые в новой грамматике могут

задавать в любой последовательности. Проверка допустимости описанных пользователем выражений производится методами валидации.

```
DptStatements
	: DptSetParentStatement
	| DptSetConditionStatement
	| DptSetPriorityStatement
;

DptSetParentStatement:
	name = 'setParent' '(' parent = [DecisionPoint|FQN] ')'
;

DptSetConditionStatement:
	name = 'setCondition' '(' (condition = ExpressionOr | 'any') ')'
;

DptSetPriorityStatement:
	name = 'setPriority' '(' (priority = ExpressionOr) ')'
;
```

Точки принятия решений типа search имеют расширенный список возможных инициализирующих высказываний.

5.6.Грамматика описания последовательностей

Грамматически последовательности разделяются на три типа: описываемые перечислением (*EnumerativeSequence*), описываемые гистограммой (*HistogramSequence*) и обычные последовательности (*RegularSequence*).

```
Sequence:
	'sequence' name = ID '=' returntype = RDOType type = SequenceType
;

SequenceType
	: EnumerativeSequence
	| HistogramSequence
	| RegularSequence
;
```

Для обычных последовательностей появилась возможность опционально задать параметры уже в момент создания последовательности. При этом задание базы генератора является обязательным.

```
RegularSequence
	: UniformSequence
	| ExponentialSequence
	| NormalSequence
	| TriangularSequence
;
```

```
UniformSequence:
      legacy ?= 'legacy'? type = 'uniform' '('
                  seed   =   INT   (','   a   =   ExpressionOr   ','   b   =
ExpressionOr)?
      ')' ';'
;
```

Полный код грамматики РДО может быть найден по следующей ссылке: *https://goo.gl/q809k0*.

6. Рабочий этап проектирования системы

На рабочем этапе проектирования системы были реализованы разработанные на предыдущих этапах схемы и концепции.

6.1.Создание ресурсов

Высказывание создания ресурса компилируется в код на языке Java классом *StatementCompiler*. Созданный ресурс регистрируется в базе данных и, в случае, если его имя не нулевое, результат записывается в локальную переменную, к которой можно будет обращаться в других конструкциях в рамках данного набора высказываний.

```
ResourceCreateStatement: {
if (localContext != null)
      localContext.addCreatedResource(st)

return
      '''
      «IF st.name != null»
            «st.reference.fullyQualifiedName» «st.name» = new «
                  st.reference.fullyQualifiedName»(«if(st.parameters  !=
null)
                        st.parameters.compileExpression.value else ""»);
            «st.name».register();
      «ELSE»
            new  «st.reference.fullyQualifiedName»(«if(st.parameters  !=
null)
                  st.parameters.compileExpression.value            else
""»).register();
      «ENDIF»
      '''
}
```

Для того, чтобы ресурс мог быть найден локально, в текущему локальному контексту добавляется его имя методом *addCreatedResource()*.

```
public void addCreatedResource(ResourceCreateStatement st) {
      for (ParameterType p : st.getReference().getParameters())
            this.addRawEntry(st.getName() + "." + p.getName(),
                  RDOExpressionCompiler.compileType(p),
                  st.getName() + ".get_" + p.getName() + "()");
}
```

В общем случае создания ресурса в наборе высказываний его регистрация в базе данных происходит без имени. Ресурсы, создаваемые в модели глобально и имеющие имена, регистрируются в базе данных по имени:

```
«rtp.name».register("«rtp.fullyQualifiedName»");
```

6.2.Глобальное описание перечислимых типов

Для каждого объявленного перечислимого типа классом *RDOEnumCompiler* генерируется класс, хранящий возможные перечислимые значения, а также описание его структуры в формате JSON.

Помимо этого, класс *RDOEnumCompiler* предоставляет статические методы *getFullEnumName(), checkValidEnumID(), compileEnumValue()* для работы с объектами перечислимого типа в выражениях.

Проверка допустимости заданного в выражении значения перечислимого типа производится методом *checkValidEnumID()*. Данный метод проверяет, является ли тип левой части выражения перечислимым, а также содержится ли переданное в правой части выражения значение в типе левой части.

```
def static boolean checkValidEnumID(String type, String id)
{
	if (!type.endsWith("_enum"))
		return false
	if (id.indexOf("(") + id.indexOf(")") != -2)
		return false
	if (!id.contains("."))
		return false

	var typeName = type.substring(type.indexOf(".") + 1)
	typeName = typeName.substring(0, typeName.lastIndexOf("."))
	val idTypeName = id.substring(0, id.lastIndexOf("."))
	if (typeName != idTypeName)
		return false

	return true
}
```

6.3.Реализация двух способов задания последовательностей

В случае, если в момент создания последовательности, ее параметры были определены, получение следующего значения может производиться как вызовом метода *next()* без параметров, так и вызовом метода *next()*, в который будут передаваться соответствующие параметры последовтаельности. В первом случае используются параметры заданные при создании последовательности, а во втором параметры переопределяются на время одного вызова. Если в момент создания последовательности ее параметры определены не были, то вызов метода *next()* без параметров запрещен.

Реализация данного подхода осуществляется генератором кода последовательностей *RDOSequenceCompiler*.

```
case "exponential": {
var ret =
```

```
        '''
        «IF (seq as ExponentialSequence).rate != null»
              private static final double rate = «
                    (seq                                                    as
ExponentialSequence).rate.compileExpression.value»;
              public static «rtype.compileTypePrimitive» next()
              {
                    return («rtype.compileTypePrimitive»)(
                              -1.0     /     rate     *     Math.log(1     -
                       prng.nextDouble()));
              }
        «ENDIF»
        '''
   ret +=
        '''
        public static «rtype.compileTypePrimitive» next(double rate)
        {
              return («rtype.compileTypePrimitive»)(
                              -1.0     /     rate     *     Math.log(1     -
                       prng.nextDouble()));
        }
        '''
   return ret
   }
```

Метод *next()* без параметров генерируется для данной последовательности только в том случае, если ее параметры уже были заданы при создании. Метод *next()* с параметрами генерируется в любом случае.

6.4.Валидация стандартных методов модели

Грамматика конструкций для переопределения стандартных методов модели была описана в максимально обобщенном виде. Чтобы пользователь получал информацию об ошибках в задании стандартных методов в процессе редактирования модели, был использован механизм валидации.

Был создан класс *DefaultMethodsHelper*, в котором для каждой грамматической конструкции, внутри которой могут переопределяться методы, были описаны допустимые имена переопределяемых методов, а также тип действия (вывод ошибки, вывод предупреждения или игнорирование) в случае, если метод не был задан.

```
   public static enum ValidatorAction {
        ERROR, WARNING, NOTHING
   };

   public static class MethodInfo {
        MethodInfo(ValidatorAction action) {
              this.action = action;
        }
```

```
        int count = 0;
        ValidatorAction action;
}
```

В классе *RDOValidator* для каждого образца, а также для корня модели выполняются проверки на корректность переопределения в них методов. Для этого предварительно формируется *HashMap*, содержаний описание всех корректных методов и счетчик числа каждого из них.

```
var Map<String, DefaultMethodsHelper.MethodInfo> counts =
            new HashMap<String, DefaultMethodsHelper.MethodInfo>()
for (v : DefaultMethodsHelper.GlobalMethodInfo.values)
        counts.put(v.name,
                    new
DefaultMethodsHelper.MethodInfo(v.validatorAction)
        )
```

После этого проводится итерирование по всем найденным внутри данной конструкции переопределениям метода и выполняются проверки на корректность его имени и отсутствие множественного переопределения. В случае обнаружения проблемы, пользователю выводится соответствующее сообщение об ошибке.

```
for(m : methods) {
        if (!counts.containsKey(m.name))
                error("Error - incorrect default method name", m,
                        m.getNameStructuralFeature
                )
        else if (counts.get(m.name).count > 0)
                error("Error - default method cannot be set more than
once", m, m.getNameStructuralFeature
                )
        else {
                var count = counts.get(m.name)
                count.count++
                counts.put(m.name, count)
        }
}
```

По окончанию данных проверок производятся дополнительные проверки для методов, которые не были переопределены ни разу. Пользователю выводится ошибка или предупреждение в зависимости от заданного в классе *DefaultMethodsHelper* действия.

7. Апробирование разработанной системы в модельных условиях

Для оценки корректности работы новой грамматики было проведено ручное сравнение результатов прогона моделей в старой грамматике и полностью соответствующих им моделей в новой грамматики.

Для оценки качества новой грамматики и сравнения ее со старой были выбраны следующие количественные характеристики:

- Размер моделей.
- Число ключевый слов.
- Число грамматических правил.

7.1.Сравнение размера моделей

Сравнение размера моделей производилось на четырех моделях:

- Модель подсчета числа звонков.
- Модель простейшей СМО.
- Модель многоканальной СМО.
- Модель пятнашек.

По результатам измерений размеров была составлена таблица:

	Модель подсчета звонков	Модель простейшей СМО	Модель многоканальной СМО	Модель пятнашек
Старая грамматика	31	62	104	137
Новая грамматика	21	52	81	88

Таблица 1 – число строк различных моделей в старой и новой грамматиках

Размер модели по данным четырем измерениями уменьшился в среднем на 27%.

7.2.Сравнение размера компонентов грамматик

По результатам измерений размеров была составлена таблица:

	Число ключевых слов	Число грамматических правил
Старая грамматика	105	145
Новая грамматика	81	132

Таблица 2 – размер компонентов старой и новой грамматик

Размер компонентов грамматики уменьшился в среднем на 15%.

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

- Была разработана новая грамматика всех основных компонентов языка РДО.
- Разработанная грамматика была описана на языке Xtext и внедрена в среду RAO-XT.
- Были проведены сравнения количественных характеристик старой и новой грамматик языка. Результаты сравнения показали ощутимое сокращение размера моделей в новой грамматике по сравнению со старой, а так же уменьшение размера самой грамматики, что свидетельствует об упрощении изучения и использования языка РДО.
- Была написана документация по новой грамматике, включающая в себя руководство по изучению языка, а также формальную справку по синтаксису языка.

Список используемых источников

1. **Емельянов В.В., Ясиновский С.И.** Введение в интеллектуальное имитационное моделирование сложных дискретных систем и процессов. Язык РДО. - М.: "Анвик", 1998. - 427 с., ил. 136.
2. **Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джеффри Д.** Компиляторы: принципы, технологии и инструментарий, 2-е изд. : Пер. с англ. - М. : ООО "И.Д.Вильямс", 2008. - 1184 с. : ил. - Парал. тит. англ.;
3. **Документация по языку РДО** [http://www.rdostudio.com/help/index.html].
4. **SWT Documentation** [https://www.eclipse.org/swt/docs.php]
5. **Xtext Documentation** [http://eclipse.org/Xtext/documentation.html]
6. **Java™ Platform, Standard Edition 8** [https://docs.oracle.com/javase/8/docs/api/]

Список использованного программного обеспечения

1. Eclipse IDE for Java Developers Luna Service Release 1 (4.4.1)
2. openjdk version "1.8.0_40-internal"
3. UMLet 13.1
4. Inkscape 0.48.4
5. Microsoft® Office Word 2010
6. Microsoft® Visio 2010
7. Railroad Diagram Generator v1.39.959

Приложение 1 – Код модели многоканальной СМО в старой грамматике

```
$Resource_type Парикмахерские: permanent
$Parameters
        количество_в_очереди: integer
$End

$Resource_type Клиенты : temporary
$Parameters
        тип      : ( Тип1, Тип2 )
        состояние: ( Пришел, Начал_стрижку )
$End

$Resource_type Парикмахеры: permanent
$Parameters
        состояние_парикмахера : ( Свободен, Занят ) = Свободен
        количество_обслуженных: integer
        длительность_min      : integer
        длительность_max      : integer
        тип_клиента           : such_as Клиенты.тип
$End

$Resources
        Парикмахерская = Парикмахерские(0);
        Парикмахер_1 = Парикмахеры(*, 0, 20, 40, Тип1);
        Парикмахер_2 = Парикмахеры(*, 0, 25, 70, Тип2);
        Парикмахер_3 = Парикмахеры(*, 0, 30, 60, Тип2);
$End

$Pattern Образец_прихода_клиента : event
$Relevant_resources
        _Парикмахерская: Парикмахерская Keep
        _Клиент        : Клиенты        Create
$Body
_Парикмахерская:
        Convert_event
                Образец_прихода_клиента.planning( time_now + Интервал_прихода( 30 ) );
                количество_в_очереди++;

_Клиент:
        Convert_event
                тип      = Тип_клиента;
                состояние = Пришел;
$End

$Pattern Образец_обслуживания_клиента : operation
$Relevant_resources
        _Парикмахерская: Парикмахерская Keep NoChange
        _Клиент        : Клиенты        Keep Erase
        _Парикмахер    : Парикмахеры    Keep Keep
$Time           =           Длительность_обслуживания(           _Парикмахер.длительность_min,
_Парикмахер.длительность_max )
$Body
_Парикмахерская:
```

```
        Choice from _Парикмахерская.количество_в_очереди > 0
        Convert_begin
                количество_в_очереди--;

_Клиент:
        Choice from _Клиент.состояние == Пришел
        Convert_begin
                состояние = Начал_стрижку;

_Парикмахер:
        Choice from _Парикмахер.состояние_парикмахера == Свободен and
_Парикмахер.тип_клиента == _Клиент.тип
        with_min( _Парикмахер.количество_обслуженных )
        Convert_begin
                состояние_парикмахера  = Занят;
        Convert_end
                состояние_парикмахера  = Свободен;
                количество_обслуженных++;
$End

$Decision_point model: some
$Condition NoCheck
$Activities
        Обслуживание_клиента: Образец_обслуживания_клиента;
$End

$Sequence Интервал_прихода : real
$Type = exponential 123456789 legacy
$End

$Sequence Длительность_обслуживания : real
$Type = uniform 123456789 legacy
$End

$Sequence Тип_клиента : such_as Клиенты.тип
$Type = by_hist 123456789 legacy
$Body
        Тип1 1.0
        Тип2 5.0
$End

$Simulation_run
        Образец_прихода_клиента.planning( time_now + Интервал_прихода( 30 ) );
        Terminate_if Time_now >= 360;
$End

$Results
        Занятость_парикмахера_1 : watch_state Парикмахер_1.состояние_парикмахера == Занят
        Занятость_парикмахера_2 : watch_state Парикмахер_2.состояние_парикмахера == Занят
        Занятость_парикмахера_3 : watch_state Парикмахер_3.состояние_парикмахера == Занят
        Обслужено_парикмахером_1: get_value Парикмахер_1.количество_обслуженных
        Обслужено_парикмахером_2: get_value Парикмахер_2.количество_обслуженных
        Обслужено_парикмахером_3: get_value Парикмахер_3.количество_обслуженных
$End
```

Приложение 2 – Код модели многоканальной СМО в новой грамматике

```
enum Тип_клиента {Тип1, Тип2}
enum Состояние_клиента {Пришел, Начал_стрижку}
enum Состояние_парикмахера {Свободен, Занят}

type Парикмахерские {
        int количество_в_очереди;
}

type Клиенты {
        Тип_клиента тип;
        Состояние_клиента состояние;
}

type Парикмахеры {
        Состояние_парикмахера состояние_парикмахера = Состояние_парикмахера.Свободен;
        int количество_обслуженных;
        int длительность_min;
        int длительность_max;
        Тип_клиента тип_клиента;
}

resource Парикмахерская = Парикмахерские.create(0);
resource Парикмахер_1 = Парикмахеры.create(*, 0, 20, 40, Тип_клиента.Тип1);
resource Парикмахер_2 = Парикмахеры.create(*, 0, 25, 70, Тип_клиента.Тип2);
resource Парикмахер_3 = Парикмахеры.create(*, 0, 30, 60, Тип_клиента.Тип2);


event Событие_прихода_клиента() {
        Клиенты.create(Случайный_тип_клиента.next(), Состояние_клиента.Пришел);
        Событие_прихода_клиента.plan(currentTime + Интервал_прихода.next());
        Парикмахерская.количество_в_очереди++;
}

operation Образец_обслуживания_клиента() {
        relevant _Парикмахерская =
Парикмахерская.select(_Парикмахерская.количество_в_очереди > 0);
        relevant _Клиент = Клиенты.select(_Клиент.состояние == Состояние_клиента.Пришел);
        relevant _Парикмахер = Парикмахеры.select(_Парикмахер.состояние_парикмахера ==
                Состояние_парикмахера.Свободен and _Парикмахер.тип_клиента ==
                 _Клиент.тип).withMin(_Парикмахер.количество_обслуженных);

        set duration() {
                return Длительность_обслуживания.next(
                        _Парикмахер.длительность_min, _Парикмахер.длительность_max);
        }

        set begin() {
                _Парикмахерская.количество_в_очереди--;
                _Клиент.состояние = Состояние_клиента.Начал_стрижку;
                _Парикмахер.состояние_парикмахера = Состояние_парикмахера.Занят;
        }
```

```
        set end() {
                _Парикмахер.состояние_парикмахера = Состояние_парикмахера.Свободен;
                _Парикмахер.количество_обслуженных++;
                _Клиент.erase();
        }
}

dpt model {
        activity Обслуживание_клиента checks Образец_обслуживания_клиента();
}

sequence Интервал_прихода = double legacy exponential(123456789, 1/30.0);
sequence Длительность_обслуживания = double uniform(123456789);
sequence Случайный_тип_клиента = Тип_клиента histogram(123456789) {
        Тип_клиента.Тип1 1.0
        Тип_клиента.Тип2 5.0
}

set init() {
        Событие_прихода_клиента.plan(currentTime + Интервал_прихода.next());
}

set terminateCondition() {
        return currentTime >= 360;
}

result Занятость_парикмахера_1 = watchState(Парикмахер_1.состояние_парикмахера == Состояние_парикмахера.Занят);
result Занятость_парикмахера_2 = watchState(Парикмахер_2.состояние_парикмахера == Состояние_парикмахера.Занят);
result Занятость_парикмахера_3 = watchState(Парикмахер_3.состояние_парикмахера == Состояние_парикмахера.Занят);
result Обслужено_парикмахером_1 = getValue(Парикмахер_1.количество_обслуженных);
result Обслужено_парикмахером_2 = getValue(Парикмахер_2.количество_обслуженных);
result Обслужено_парикмахером_3 = getValue(Парикмахер_3.количество_обслуженных);
```