Волков Г. В. ИУ7-81Б

РЕФЕРАТ

Расчётно–пояснительная записка 42 с., 5 рис., 1 табл., 12 источн., 1 прил.

МОДЕЛИРОВАНИЕ, МНОГОФУНКЦИОНАЛЬНЫЕ ЦЕНТРЫ ОБСЛУЖИВАНИЯ, СЕТИ ПЕТРИ, КОНЕЧНЫЕ АВТОМАТЫ, ВЕРОЯТНОСТНЫЕ АВТОМАТЫ

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Моделирование — процесс замещения одного объекта другим с целью получения информации о важнейших свойствах объекта–оригинала с помощью объекта–модели. Широко применяется в научных исследования и в прикладных задачах в различных областях. Компьютерное моделирование является одним из эффективных методов изучения сложных систем. Модели проще и удобнее исследовать, когда реальные эксперименты затруднены из–за финансовых или физических препятствий. Формализованность позволяет чётко обозначить основные факторы, определяющие свойства изучаемого объекта–оригинала и связи между ними.

В современном мире управление операционной деятельностью и оптимизация процессов обслуживания стали ключевыми вопросами для организаций, предоставляющих различные виды услуг. Одной из важных составляющих этой области является моделирование многофункциональных центров обслуживания, пересекающихся с различными видами услуг и комплексными процессами. Классификация методов моделирования многофункциональных центров обслуживания и анализ их применимости представляют высокую актуальность для исследователей и практиков.

Моделирование многофункциональных центров позволяет проанализировать и проконтролировать правильность функционирования систем, без больших затрат на оборудование, персонал и обслуживание. Также даёт возможность обнаружить ошибки проектирования на этапе подготовки, а не во время эксплуатации, что также значительно снижает расходы.

Актуальность этой темы объясняется большим и постоянно растущем спросом людей на услуги предоставляемые данными центрами. Только за 2022 в Москве в таких центрах было оказано более 27 миллионов услуг [2]. Для анализа и контроля правильности функционирования многофункциональных центров обслуживания применяется моделирование. Это позволяет сокращать время проектирования, уменьшать конечную стоимость создания центров, исключаем множественные исправления дефектов выявленных в ходе эксплуатации.

Цель данной работы — разработка метода моделирование многофункциональных центров обслуживания клиентов на основе сетей Петри, разра-

ботка программного обеспечения.

Для достижения поставленной в работе цели предстоит решить следующие задачи:

— изучить основные понятия моделирования многофункциональных центров обслуживания, описать и сравнить существующие формализмы и методы протяжки модельного времени;

— формализовать постановку задачи;

— разработать модель функционирования многофункциональных центров и моделирующий алгоритм;

— реализовать разработанный метод;

— провести исследование эффективности реализованного метода для разных конфигураций многофункциональных центров и сравнить с GPSS World;

1 Аналитический раздел

1.1 Анализ предметной области

1.1.1 Основные понятия

При моделировании важно использовать модель, адекватную исследуемой системе. Это означает, что существенные с точки зрения разработчика свойства модели и системы в достаточной для анализа степени должны совпадать. В исследованиях используют модели, а не реальные системы по следующим причинам: реальные системы очень сложны, поэтому для их анализа применяются упрощённые модели, или проведение эксперимента просто невозможно, из–за каких–либо физических ограничений. Определения основных понятий [1]:

— система — совокупность объектов, взаимодействующих друг с другом, которая может является частью другой системы и включать в себя системы;

— модель — объект, созданный для получения новых знаний о объекте–оригинале, отражающий только существенные свойства оригинала;

— моделирование — исследование каких–либо явлений, систем или процессов путём построения и анализа модели.

На рисунке 1.1 представленная классификация основных видов моделирования [1].

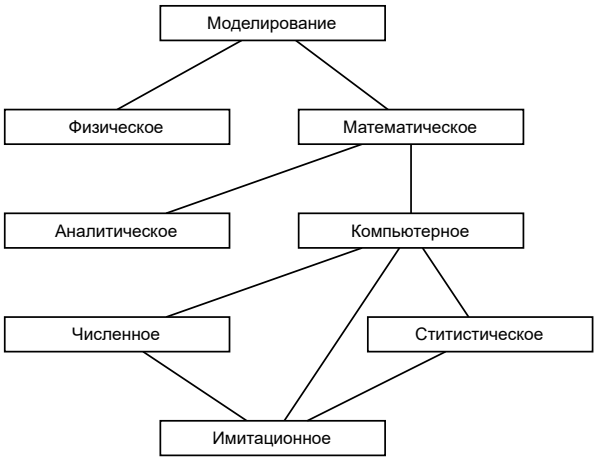

Рисунок 1.1 – Виды моделирования

При **физическом** моделировании используется сама система или подобная ей. Физическая модель может быть реализована в уменьшенном или увеличенном масштабе [1].

Под **математическим** моделированием понимается процесс установления соответствия реальной системе математической модели и исследование этой модели, позволяющее получить характеристики реальной системы. Применение математического моделирования позволяет исследовать объекты, реальные эксперименты над которыми затруднены или невозможны. В зависимости от вида модели математическое моделирование делится на аналитическое и компьютерное. Заметим, что аналитическое решение предпочтительнее, но его не всегда удаётся получить [1].

При **компьютерном** моделировании модель формулируется в виде алгоритма или программы. Можно разделить на численное, статистическое и имитационное [1].

При **численном** моделировании используются методы вычислительной математики [1].

При **статистическом** моделировании выполняется обработка данных о системе с целью получения статистических характеристик системы [1].

При **имитационном** моделировании процесс функционирования исследуемой системы воспроизводится на ЭВМ при соблюдении логической и временной последовательности протекания процессов, что позволяет узнать данные о состоянии системы или отдельных ее элементов в определённые моменты времени [1].

1.1.2 Моделирование многофункциональных центров обслуживания

Многофункциональные центры обслуживания (МФЦ) — это современные организации, призванные обеспечивать широкий спектр административных и государственных услуг гражданам и юридическим лицам в одном месте. В последние годы подобные центры получили широкое распространение в многих странах, в том числе и в России, где они известны как «Мои документы». Являются структурированной системой, предназначенной для предоставления различных видов услуг клиентам. Они могут включать в себя комплексные процессы, включающие как прямое обслуживание клиентов, так и внутренние операционные процессы.

Для эффективного управления многофункциональными центрами обслуживания необходимо иметь понимание их работы и оптимальные стратегии управления. Моделирование является мощным инструментом, который позволяет анализировать и прогнозировать процессы обслуживания, а также оптимизировать их эффективность.

Услуги в центре могут оказываться как непосредственно представители организаций–участников, так и универсальными специалистами, являющимися работниками центра. Помимо этого все услуги, оказываемые на площадке МФЦ, можно разделить на три типа [3]:

— консультации (результатом таких услуг является информация, за которой прошёл заявитель);

— приём документов (при получении таких услуг заявить приносит и отдаёт некий набор документов);

— выдача документов (как правило за такими услугами обращаются после первых двух, при их получении заявителю передаётся некоторый набор бумаг).

Разные типы услуг стоит рассматривать по разному в процессе моделирования.

В результате моделирования центра можно получить множество различных параметров его работы, на основе которых делать выводы об текущей

эффективности и предлагать улучшения, которые также можно будет промоделировать. Такой итеративный процесс позволит создать экономичную и эффективную систему обслуживания клиентов. Выделяются следующие характеристики, имеющие практические ценность, центра, которые можно получить в результате моделирования [4], [5]:

— среднее время обслуживания;

— среднее время пребывания клиентов в очереди;

— вероятность простоя специалиста;

— вероятность попадания клиента в очередь;

— вероятность ухода клиента.

Исходя из этих и других характеристик можно оценить общую эффективность работы, экономическую и социальную эффективность и т. д. Все характеристики измеряются отдельно для разных типов специалистов и очередей.

1.2 Классификация методов моделирования многофункциональных центров обслуживания

Рассмотрим наиболее общие и часто используемые методы, применяемые для моделирования многофункциональных центров обслуживания.

1.2.1 Конечные автоматы

Автомат можно представить как некоторое устройство (чёрный ящик), на которое подаются входные сигналы, снимаются выходные сигналы и которое может иметь определённые внутренние состояния. Они являются дискретно–детерминированными моделями (F–схема) [6].

Введём понятие алфавит, понимая под ним конечное множество объектов любой природы. В этом случае сами объекты можно называть буквами, ах конечную упорядоченную совокупность называют словом [6].

Конечный автомат имеет один вход и один выход. Он представляет собой объект, функционирующий в дискретные моменты времени. В каждый момент времени t_i автомат находится в одном из возможных состояний $z(t_i)$. Начиная с нулевого момента времени на вход автомата поступает входной сигнал , который является одной из букв входного алфавита . Автомат следующим образом реагирует на поступление входных сигналов. Во–первых, состояние автомата изменяется в соответствии с одношаговой функцией переходов:

$$z(t_i) = \varphi(z(t_{i-1}), x(t_i)). \tag{1.1}$$

Во–вторых в каждый момент на выходе автомата появляется выходной сигнал $y(t_i)$, который является буквой выходного алфавита Y, и определяется функцией выходов:

$$y(t_i) = \psi(z(t_{i-1}), x(t_i)). \tag{1.2}$$

Таким образом конечный автомат можно определить как кортеж $A = (X, Y, Z, z_0, \varphi, \psi)$, где $X = \{x_1, ..., x_m\}$ — множество входных сигналов (входной алфавит), $Y = \{y_1, ..., y_n\}$ — множество выходных сигналов (выходной алфавит), $Z = \{z_1, ..., z_f\}$ — множество состояний (внутренний алфавит), z_0 — начальное состояние, φ — функция переходов, которая некоторым парам «состояние – входной сигнал» ставит в соответствие новое состояние автомата, ψ — функция выходов, которая некоторым парам «состояние – входной сигнал» ставит в соответствие выходные сигналы автомата. В общем случае конечный автомат может иметь много входов, состояний и выходов. В этом случае алфавиты представляют собой прямые произведения более простых алфавитов [6].

Смысл работы автомата состоит в том, что он реализует некоторое отображение множества слов входного алфавита во множество слов выходного алфавита . На уровне абстрактной теории понятие «работа автомата» понимается как преобразование входных слов в выходные [6].

На практике наибольшее распространение получили автоматы Мили и Мура. Автомат Мили функционирует по формулам (1.1) и (1.2), то есть

состояние и выходной сигнал зависят от входного сигнала и предыдущего состояния. У автомата Мура функция переходов совпадает с формулой (1.1), но функция выходов имеет вид $y(t_i) = \psi(z(t_{i-1}))$, то есть не зависит от входного сигнала. Автомат Мили — более общий автомат, чем автомат Мура. У каждого конечного автомата Мура есть конечный автомат Мили его интерпретирующий [7].

В табличном виде автомат Мили задаётся двумя таблицами. Первая таблица описывает функцию перехода, в ней столбцы это состояния, а строки входные символы, а на пересечении находятся новые состояния. Вторая таблица аналогичным образом описывает функцию выходов. Для автомата Мура первая таблица такая же, а вторая состоит только из двух строк состоянии и выходов [7].

При решении задач моделирования часто более удобной формой является матричное задание конечного автомата. При этом можно рассматривать две матрицы — матрицу переходов и матрицу выходов. Матрица переходов есть квадратная матрица, строки которой соответствуют исходным состояниям, а столбцы - состояниям перехода. Элементы на пересечении соответствует входному сигналу, вызывающему переход. Матрица выходов строится аналогично, но ее элемент соответствует выходному сигналу, выдаваемому при переходе. При матричном задании конечного автомата Мура матрица переходов аналогична соответствующей матрице автомата Мили, а выход описывается вектором выходов [7].

Ещё есть графический способ, при котором автомат представляется в виде направленного графа. Вершинами являются состояния, если из одного состояния можно перейти в другое, то они соединяются направленной дугой и ей присваивается метка. Эта метка содержит входной и выходной символ перехода [7]. Граф автомата Мили изображён на рисунке **??**.

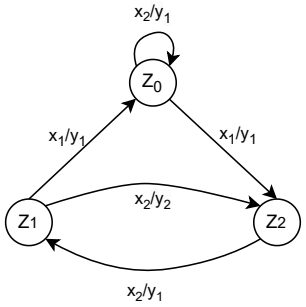

Рисунок 1.2 – Граф автомата Мили

Граф автомата Мура задаётся аналогично, только метка пути не содержит выходного символа. Он привязан к вершине [7]. Граф автомата Мура изображён на рисунке **??**.

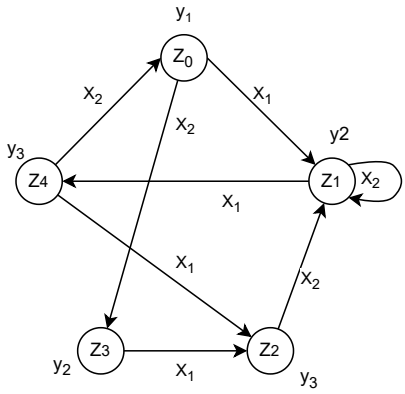

Рисунок 1.3 – Граф автомата Мили

По характеру отсчёта времени конечные автоматы делятся на синхронные и асинхронные. Автомат считается синхронным, когда моменты поступления входных сигналов, изменения состояний и выдачи выходных сигналов, определяются принудительно синхронизирующими сигналами (заранее определены). Реакция автомата на каждое значение входного сигнала заканчивается за один такт синхронизации. Асинхронные автоматы не имеют «жесткой» тактности. Они изменяют свои состояния при поступлении входных сигналов, которые могут появляться в произвольные моменты времени из некоторого интервала [6].

Конечные автоматы работают с дискретным временем и позволяют моделировать только детерминированные объекты. Также они имеют, только одно состояние на всю систему и не способны отражать параллельные процессы.

1.2.2 Вероятностные автоматы

Дискретно–стохастический подход (Р–схемы) использует в качестве математического аппарата вероятностные автоматы, которые можно определить, как дискретные потактные преобразователи информации с памятью, функционирование которых в каждом такте зависит только от состояния памяти в них и может быть описано статистически. Для такого автомата харак-

терно задание таблицы вероятностей перехода автомата в некоторое состояние и появления некоторого выходного сигнала в зависимости от текущего состояния и входного сигнала [8].

Конечный автомат рассматривался как детерминированный, то есть каждой паре состояния и входа однозначно сопоставлялись новое состояние и выход с помощью функций переходов и выходов. Вероятностный конечный автомат — такой автомат, который вместо однозначного соответствия задаёт лишь условные вероятности появления некоторых пар «новое состояние – выход», при условии реализаций некой пары «состояние – вход» [9].

Кроме того, для вероятностного конечного автомата не задаётся однозначно начальное состояние, а задаётся лишь безусловные вероятности, с которыми каждое из состояний может оказаться начальным. Эти вероятности должны быть в промежутке от 0 до 1 и в сумме давать 1 [9].

Описание функционирования вероятностного конечного автомата можно трактовать так, что для каждой пары «состояние – вход» задаётся совместное условное распределение вероятностей осуществления пар «состояние – выход». Сумма этих вероятностей должна быть равна единице для каждой пары «состояние – вход» [9].

Если считать вероятности нового состояния и выхода независимо друг от друга, то этот автомат называется вероятностным автоматом Мили. Также подобно конечному автомату Мура возможен вероятностный конечный автомат, у которого выход не зависит от входа, а зависит только от текущего состояния [9].

Вероятностный конечный автомат называется автономным, если все случайные законы распределения одинаковы при различных вариантах сочетания входных алфавитов [9].

Вероятностные автоматы работают аналогично конечным автоматам, но позволяют моделировать стохастические системы.

1.2.3 Системы массового обслуживания

Непрерывно–стохастический подход (Q–схема) применяется для формализации процессов обслуживания. Этот подход наиболее известен ввиду того, что большинство производственных, экономических, технических и т.д.

систем по сути являются системами массового обслуживания. Под системой массового обслуживания понимают динамическую систему, предназначенную для эффективного обслуживания потока заявок при ограничениях на доступные ресурсы. В любой системе массового обслуживания можно выделить элементарный прибор, в котором уже выделяют накопитель заявок некоторой ёмкости, ожидающих обслуживания, канал обслуживания и потоки событий. Существует поток заявок на обслуживание, характеризующийся моментами времени поступления и их атрибутами, и поток обслуживания, характеризующийся моментами начала и окончания обслуживания заявок. Под непрерывностью тут обозначается непрерывность времени. Моменты поступления заявки в систему и окончания обслуживания заявки — случайны [8]. Структура системы массового обслуживания приведена на рисунке 1.4.

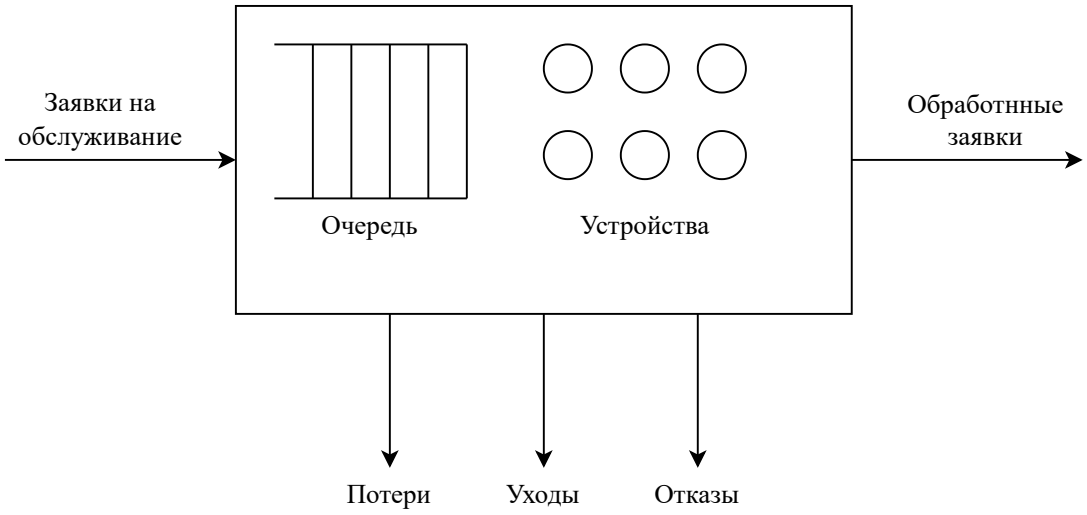

Рисунок 1.4 – Стрктурная схема СМО

Основными задачами решаемые в рамках теории массового обслуживания являются: анализ, то есть определение количественных характеристик СМО при заданной структуре и параметрах элементов, и синтез оптимальной структуры при заданных характеристиках и ограничениях на параметры элементов [6].

На вход СМО поступают заявки на обслуживание, образующие входящий поток. Они поступают из источником заявок. В зависимости от характера источника заявок различают разомкнутые и замкнутые СМО. В разомкнутых число заявок, вырабатываемых источником, считается неограниченным и поведение источника никак не связано с состоянием системы в любой момент времени. Для замкнутых СМО характерно конечное количество заявок,

циркулирующих в системе. Обслуженные заявки возвращаются в источник и через некоторый момент времени могут попасть опять на вход [6].

По характеру обслуживания заявок все системы массового обслуживания делятся на три типа:

— системы с отказами;

— системы с ожиданием;

— системы смешанного типа.

СМО первого типа характеризуются тем, что поступившие в данный момент времени заявка тут же принимается к обслуживанию, если имеется хотя бы один свободный канал, или получает отказ, если все каналы заняты. В системах с ожиданием заявка в случае занятости всех каналов обслуживания становится в очередь и ожидает освобождения одного из них. Это наиболее представительный на практике класс систем. Системам смешанного типа присущи особенности двух вышеописанных систем. Заявка в такой системе становится в очередь, если в ожидании находится ограниченное число заявок. Этим числом может быть длинна очереди. Ограничение также может быть и по длительности ожидания [10].

По дисциплине обслуживания заявок выделяют системы без приоритета и с приоритетом [10].

Ещё одним из признаков классификации является количество обслуживающих устройств или каналов: если система имеет один прибор или один канал, то она называется одноканальной, если же их более одного, то она называется многоканальной [11].

Важнейшей характеристикой СМО является входящий поток заявок. Число заявок в единицу времени, обычно называется интенсивностью и обозначается λ. Если входящий поток является простейшим, поток распределён по закону Пуасона, то достаточно знать лишь λ (или интервал поступление заявок $t_i = \frac{1}{\lambda}$). А интервал входящего на обслуживание потока $v_\lambda = 1$. В общем случает надо знать среднее значение λ и t_i и дисперсионную характеристику интервала входящего на обслуживание потока v_λ [10].

Система характеризуется числом каналов обслуживания n, длительностью обслуживания t_p одной заявки и пропускной способностью μ, число заявок, которое может обслужить поток в единицу времени. Отношение

$\rho = \frac{\lambda}{\mu} = \frac{t_p}{t_i}$ называется коэффициентом использования пропускной способности или приведённой плотностью потока [10].

При работе система имеет одно состояние, которое меняется в зависимости от количества занятых каналов.

Многофункциональный центр обслуживания можно смоделировать используя, разомкнутую многоканальную СМО смешанного типа без приоритетов. Тогда рассмотрим n–канальную СМО с ожиданием, на которую поступает поток заявок с интенсивностью λ. Интенсивность обслуживания одного канала m. Число мест в очереди m. Тогда основные показатели работы СМО приведены ниже [10].

Вероятность того, что система находится в состоянии, в котором все каналы свободны обознается как p_0 и вычисляется по формуле

$$p_0 = (\sum_{i=0}^{n} \frac{\rho^i}{i!} + \frac{\rho^n + 1(1 - (\frac{\rho}{n})^m)}{n \cdot n!(1 - \frac{\rho}{n})})^{-1}. \tag{1.3}$$

Среднее число заявок в очереди обознается как L_q и вычисляется по формуле

$$L_q = \frac{\rho^{n+1} p_0 \left(1 - \left(m + 1 - m\frac{\rho}{n}\right)\left(\frac{\rho}{n}\right)^m\right)}{n \cdot n! \left(1 - \frac{\rho}{n}\right)^2}. \tag{1.4}$$

Среднее число заявок под обслуживанием или среднее число занятых каналов обознается как $\bar{k}$ и вычисляется по формуле

$$\bar{k} = \rho \left(1 - \frac{\rho^{n+m}}{n^m \cdot n!} p_0\right). \tag{1.5}$$

Среднее число заявок в системе обознается как L_s и вычисляется по формуле

$$L_s = L_q + \bar{k}. \tag{1.6}$$

Относительная пропускная способность обознается как Q и вычисляется по формуле

$$Q = 1 - \frac{\rho^{n+m}}{n^m n!} p_0. \tag{1.7}$$

Абсолютная пропускная способность системы обознается как A и вычисля-

ется по формуле

$$A = \lambda Q. \tag{1.8}$$

Под системой массового обслуживания понимают динамическую систему, предназначенную для эффективного обслуживания потока заявок при ограничениях на ресурсы системы. Она является непрерывно–стохастической моделью, где непрерывность подразумевает непрерывность времени. Позволяет моделировать случайные процессы. К данной модели сводится множество систем реального мира. Данный метод позволяет моделировать стохастические и параллельные системы, но имеет только одно состояние на всю систему, отображающее количество занятых потоков.

1.2.4 Сети Петри

Сетевой подход (N–схема) используется для формализованного описания и анализа причинно–следственных связей в сложных системах, где одновременно протекает несколько процессов. Самым распространенным формализмом, описывающим структуру и взаимодействие параллельных систем и процессов, являются сети Петри [8].

Сеть Петри — это математическая модель дискретных динамических систем, ориентированная на качественный анализ и синтез таких систем. Формально в терминах теории систем сеть Петри это кортеж

$$PN = (O, P, T, F, M_0), \tag{1.9}$$

где:

— $O = \{0, 1, 2, \ldots\}$ — множество дискретных моментов времени;

— $P = \{p_1, p_2, \ldots\}$ — непустое множество элементов сети, называемых позициями;

— $T = \{t_1, t_2, \ldots\}$ — непустое множество элементов сети, называемых переходами;

— $F : (P \times T) \cup (T \times P) \to \{0, 1, 2, \ldots, k, \ldots\}$ — функция инцидентности, где k — кратность дуги;

— M_0 — начальная маркировка позиций.

Множества позиций и переходов не пересекаются [12].

Функция инцидентности может быть представлена в виде $F = F^p \cup F^t$ и фактически задаёт два отображения: $F^p(p,t) = P \times T \to \{0,1,2,\ldots\}$, т.е. для каждой позиции указываются связанные с ней переходы (с учётом их кратности); $F^t(t,p) = T \times P \to \{0,1,2,\ldots\}$, т.е. для каждого перехода указываются связанные с ним позиции (также с учётом кратности) [12].

Эти функции, в общем случае зависящие от времени, могут быть представлены матрицами инцидентности. Из вершины–позиции $p_i \in P$ дуга в вершину–переход $t_j \in T$ существует тогда и только тогда, когда элемент на пересечении i–ой строки и j–го столбца в матрице $f^p_{ij} > 0$. В этом случае говорят, что t_i выходной переход позиции p_i. Аналогичным образом определяется выходная позиция перехода [12].

Каждая позиция $p_i \in P$ может содержать некоторый целочисленный ресурс $\mu(p) \geq 0$, называемый числом фишек внутри позиции. Вектор $M = [\mu_1, \mu_2, \ldots]$ называется маркировкой (разметкой) сети Петри. Каждая маркировка — это отображение $M : P \to \{0,1,2,\ldots\}$ [12].

Сети Петри функционирует в дискретном времени и концентрируют внимание на локальных событиях (переходах), локальных условиях (позициях) и локальных связях [12].

Смена маркировок (начиная с M_0) происходит в результате срабатывания переходов сети. Переход сети $t_j \in T$ может сработать при маркировке M, если для всех входных позиций $p_i \in P$ выполняется условие $\mu(p_i) - f^p_{ij} \geq 0$, т.е. если каждая входная позиция для данного перехода содержит столько же и ли больше фишек чем кратность ведущей к переходу дуги. В результате срабатывания перехода в момент времени θ происходит смена маркировки по правилу: $\mu_i(\theta + 1) = \mu_i(\theta) - f^p_{ij}(\theta) - f^t_{ji}(\theta)$. То есть переход изымает из каждой своей входной позиции число фишек, равное кратности входных дуг, и посылает в каждую свою выходную позицию число фишек, равное кратности выходных дуг. Если может сработать несколько переходов, то срабатывает один, любой из них. Функционирование сети останавливается, если при некоторой маркировке ни один из ее переходов не может сработать. В силу своей недетерминированности при одинаковой начальной разметки сети Петри могут порождать различные последовательности срабатывания ее пе-

реходов. Эти последовательности образуют слова в алфавите T. Множество всевозможных слов, порождаемых сетью Петри, называют языком сети Петри. Две сети Петри эквивалентны, если порождают один и тот же язык [12].

Сети Петри также представимы в виде двудольного ориентированного мультиграфа. Этот граф содержит:

— позиции (места), обозначаемые кружками;

— переходы, обозначаемые планками;

— ориентированные дуги (стрелки), соединяющие позиции с переходами и переходы с позициями.

Благодаря наличию кратных дуг сеть Петри есть мультиграф. Благодаря двум типам вершин граф называется двудольным. Поскольку дуги имеют направление, граф является ориентированным [12]. Пример такого графа изображён на рисунке 1.5.

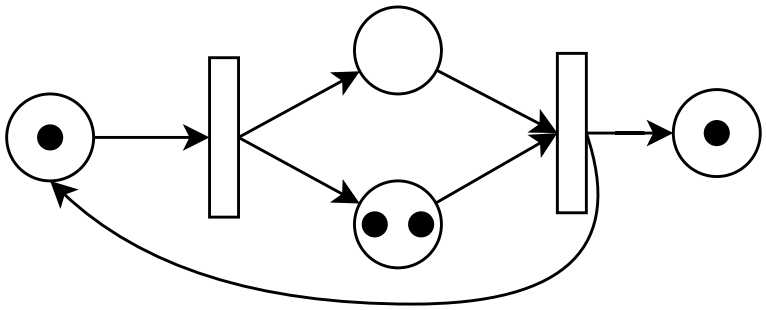

Рисунок 1.5 – Граф сети Петри

Свойства сетей Петри

Исходя из практических задач моделирования, можно установить ряд свойств сетей Петри, характеризующих поведение моделируемых систем [12].

Свойство ограниченности. Позиция p_i в сети называется ограниченной если для любой достижимой в сети маркировки M существует такое k, что $\mu_i \leq k$. Сеть называется ограниченной, если все ее позиции ограничены.

Свойство безопасности. Сеть называется безопасной, если при любой достижимой маркировке для любого $\mu_i \leq 1$. Таким образом, в безопасной сети вектор маркировок состоит только из нулей и единиц.

Свойство консервативности. Сеть называется консервативной, если сумма фишек во всех позициях остаётся постоянной при работе сети.

Свойство живости. Переход t_i называется потенциально живым, если существует достижимая из M_0 маркировка M', при которой t_i может сработать. Если t_i является потенциально живым при любой достижимой маркировке, то он называется живым. Переход t_i, не являющийся потенциально живым при M_0 называется мёртвым при этой маркировке. Маркировка M_0 в этом случае называется тупиковой для t_i. Переход называется устойчивым, если никакой другой переход не может лишить его возможности сработать при наличии для этого необходимых условий, то есть нет конфликта.

Модификации сетей Петри

Базовое определение сети Петри позволяют моделировать широкий класс дискретных систем. Однако в ряде случаев этих возможностей оказывается недостаточно, поэтому вводят обобщения этих сетей, которые обладают расширенными возможностями моделирования [12].

Ингибиторные сети — это сети Петри, для которых функция инцидентности имеет вид $F = F^p \cup F^t \cup F^i$, т.е. она дополнена специальной функцией инцидентности $F^i(t,p) = T \times P \to \{0,1\}$, которая вводит ингибиторные дуги. Правила срабатывания переходов в ингибиторной сети модифицируются следующим образом. Переход t_i срабатывает, если для всех связанных с ним позиций p_i и p_k выполняется $(\mu(p_i) \geq f^p_{ij}) \wedge (\mu(p_k) \cdot f^i_{kj} = 0)$, где p_k позиция связанная с переходом ингибиторной дугой. То есть позиции связанные через ингибиторные дуги не должны содержать фишек.

Сети с приоритетами — это сети Петри, в которой каждому переходу приписан приоритет. В стандартной сети Петри в случае, когда могут сработать несколько переходов, срабатывает любой из них. При моделировании реальных систем могут сложиться ситуации, когда последовательность срабатываний необходимо регламентировать. Это можно сделать, введя множество приоритетов $PR : T \to \{0, 1, \dots\}$ и приписав каждому из переходов t_j соответствующее целочисленное значение приоритета $pr_j s$. Тогда правило срабатывания переходов модифицируется: если на некотором такте работы сети PN имеется возможность для срабатывания нескольких переходов, то срабатывает тот из них, который имеет наивысший приоритет.

Сети со случайными срабатываниями переходов – это сети Петри, в которой каждому переходу t_i приписана вероятность срабатывания p_i, при этом $\sum_{i=0}^{n} p_i = 1$, где n — количество переходов. То есть в случае, когда могут сработать несколько переходов, они срабатывают с учётом своих вероятностей и для каждой группы таких переходов сумма их вероятностей должна равняться единице. Отождествив маркировки с состоянием сети и положив, что вероятности не зависят от работы сети в предыдущие такты, мы получим цепь Маркова, описывающие вероятностное поведение системы.

Иерархические сети Петри представляют собой многоуровневые структуры, в которых выделяются сети различного уровня. Они позволяют моделировать различные многоуровневые (иерархические) системы. В отличие от обыкновенных сетей Петри, в иерархических сетях имеются два типа переходов: простые и составные. Составные переходы содержат внутри себя сеть Петри более низкого уровня. Формально они состоят из входного и выходного переходов, между ними находится некоторая сеть Петри, которая, в свою очередь, также может быть иерархической. Срабатывание составных переходов является не мгновенным событием, а составным действием. На каждом шаге дискретного времени составной переход может находиться в одном из двух состояний - пассивном и активном. Составной переход может быть активирован , если он до этого был пассивен и имеются условия для срабатывания его головного перехода. При этом производится изменение маркировки в сети верхнего уровня по обычным правилам и запускается работа в сети, находящейся внутри составного перехода. Сеть нижнего уровня работает с учётом своей начальной маркировки до тез пор, пока все ее переходы не станут пассивными, т.е. не смогут сработать. После этого происходит срабатывание хвостового перехода и изменение маркировки сети верхнего уровня. Составной переход возвращается в пассивное состояние, а в сети нижнего уровня восстанавливается начальная маркировка.

Временные сети Петри называют пару (PN, f), где PN — ординарная сеть Петри, а f – функция задержки переходов t, $f : T \to N^+$, где N^+ — множество целых неотрицательных чисел. Таким образом, каждому переходу t_i сети Петри приписывается некоторое число $\tau_i = f(t_i)$, смысл которого состоит в следующем. После момента времени, когда переход оказался возбуждённым, он срабатывает ровно через τ_i единиц времени. Если при данной

маркировке несколько переходов конфликтуют, то срабатывает тот из них, задержка которого минимальна. Если при данной маркировке возбуждены два или более перехода с одинаковой задержкой, то они сработают одновременно [13].

Стохастические сети Петри — отличается от ординарной сети тем, что в ней каждый переход помечается либо средней скоростью его срабатывания (время срабатывания возбуждённого перехода — случайная величина), либо вероятностью его срабатывания при возбуждении, в то время как в ординарной сети рассматривается лишь возможность срабатывания возбуждённого перехода. Формально стохастическая сеть Петри представляет собой расширение ординарной СП, за счёт введения дополнительного отображения $\lambda : T \to N^+$, где N^+ — множество целых неотрицательных чисел. Иначе говоря, каждому переходу $t_i \in T$ отображение λ ставит в соответствие некоторое число λ_i, интерпретируемое как средняя скорость срабатывания этого перехода. Эта характеристика носит вероятностный характер. Обычно полагают, что скорость срабатывания перехода имеет показательное распределение. Имеет место теорема о том, что любая конечная стохастические сети Петри изоморфна одномерной дискретной марковской цепи [13].

Цветные сети Петри (CPN). Для CPN не существует чёткого определения. CPN являются всего лишь некоторым абстрактным термином, обозначающим сети Петри с расширенными возможностями, как минимум, в отношении типизации элементов сетей. Наиболее распространённой и общей является версия разработанная Куртом Иенсеном (университет Орхуса) [14].

Типы в CPN называются наборами цветов и включают целый, логический, интервалы целых, конечные перечисления, а также структурные типы — кортежи, записи и списки ограниченной или неограниченной длины. Для перехода к цветной сети Петри, надо задать совокупность типов данных (наборов цветов) Σ и функцию цветности $C : P \to \Sigma$, которая каждому месту p приписывает некоторый тип $C(p)$. Тип места указывает тип значений, которые может содержать фишка, находящийся в этом месте. А какие именно фишки и в каком количестве находятся в каждом месте задаёт разметка. То есть, разметка M это функция, которая с каждым местом связывает мультимножество $M(p) \in C(p)_{MS}$. Далее, каждой дуге $a \in A$ задаётся выражение $E(a)$ типа $C(p)_{MS}$, где p — место, связанное с дугой a. Для каждого перехода

t определяется выражение $G(t)$ логического типа – охрана (по умолчанию – истина) [15].

Все выражения могут содержать в качестве свободных переменных только переменные из заранее заданного списка переменных V , причём для каждой переменной $v \in V$ должен быть зафиксирован ее тип $Type(v) \in \Sigma$. Если задано связывание $b : V \to \cup\Sigma$ всех переменных $v \in V$ со значениями $b(v)$ нужного типа $Type(v)$, то всякое выражение $G(t)$ или $E(a)$ может быть вычислено до значения логического типа или $C(p)_{MS}$, соответственно. Для результата вычисления будем использовать обозначение $E(a)\langle b\rangle$, соответственно $G(t)\langle b\rangle$. При этом связывание b должно содержать как минимум все (свободные) переменные данного выражения. В качестве начальной разметки M_0 для каждого места p задаётся замкнутое (не содержащее свободных переменных) выражение $I(p)$, которое вычисляется до некоторого конечного мультимножества типа $C(p)_{MS}$. Таким образом, цветная сеть Петри задаётся как набор из десяти компонентов: $CPN = (P, T, A, N, \Sigma, V, C, E, G, I)$ [15].

Внешний токен — это пара (p, v), где $v \in C(p)$. Множество всех возможных внешних токенов данной CPN обозначим TE. Внешнее связывание это пара (t, b), где $t \in T$ и $b \in B(t)$. Множество всех внешних связываний перехода t определим как $BE(t) = (b, t)|b \in B(t)$. Множество всех внешних связываний данной CPN обозначим BE. Шаг $Y \in BE_{MS}$ это непустое конечное мультимножество внешних связываний. Определим $A(p, t)$ как множество всех дуг из p в t. $A(p, t) = a|N(a) = (p, t)$. Аналогично $A(t, p)$. Определим $E(p, t)$ как формальную сумму всех выражений на дугах из $A(p, t)$. Это допустимо, поскольку типы значение всех этих выражений одинаковые. Если нет дуг из p в t, то $E(p, t) = \varnothing$. Аналогично $E(t, p)$ [15].

Связывание (t, b) допускается разметкой M_1, если для всякого места p выполняется $E(p, t)\langle b\rangle \leq M_1(p)$. Тогда переход t (со связыванием b) может сработать, породив новую разметку $M2 = M_1 - \Sigma_{p\in P}E(p, t)\langle b\rangle + \Sigma_{p\in P}E(t, p)\langle b\rangle$. Иначе говоря, значения выражений $E(p, t)\langle b\rangle$ на входных дугах показывают мультимножества токенов, которые необходимы для срабатывания перехода t со связыванием b и которые при этом срабатывании будут из разметки удалены, а взамен будет добавлено мультимножество токенов, заданное выражением $E(t, p)\langle b\rangle$ [15].

Шаг Y допускается разметкой M_1, если для всякого места p справед-

ливо $\Sigma_{(t,b)\in Y} E(p,t) \leq M_1(p)$, где суммирование производится с учётом кратностей элементов Y. Тогда могут сработать одновременно все элементы шага Y, порождая новую разметку M_2, такую что для всякого места p выполняется $M_2(p) = M_1(p) - \Sigma_{(t,b)\in Y} E(p,t) + \Sigma_{(t,b)\in Y} E(t,p)$io . И тогда говорят, что разметка M_2 непосредственно достижима из разметки M_1. Конечная или бесконечная последовательность шагов, а также отношение достижимости определяются и обозначаются так же, как и для стандартных сетей Петри [15].

Вывод

Сеть Петри — это математическая модель дискретных динамических систем, ориентированная на моделирование параллельных систем. Имеет множество обобщений, расширяющих её функционал. Позволяет моделировать стохастические системы. Сконцентрирована на локальных событиях в системе, что позволяет отображать состояние всей системы и ей отдельных элементов. Позволят выделять некоторые части в отдельные функциональные блоки с помощью иерархических сетей. Также стохастические сети Петри позволяют учитывать, не только время, но и его вероятностные параметры, что особенно важно для моделирования МФЦ. Цветные сети Петри позволяют разграничить заявки по типам, а также собирать статистику отдельно по каждой заявке, а не только по элементам системы.

1.2.5 Сравнение методов моделирования многофункциональных центров обслуживания

Для рассмотренных методов выделим следующие критерии сравнения:

— возможность моделирования стохастических систем (К1);

— возможность моделирования параллельных систем (К2);

— тип состояния (К3) — некоторые методы имеют только одно глобальное состояние на всю системы, другие концентрируются на локальных событиях, условиях и связях, что позволяет получить более подробную информацию о состоянии всей системы и её отдельных элементов;

— учёт времени (К4) — возможность учёта времени при моделировании;

— учёт времени (К5) — возможность учёта различных типов заявок в системе.

Результаты сравнение приведённых методов по выделенным критериям представлены в таблице 1.1.

Таблица 1.1 – Результаты сравнения методов

Критерий	КА	ВА	СМО	СП
К1	Нет	Да	Да	Да
К2	Нет	Нет	Да	Да
К3	Глобальное	Глобальное	Глобальное	Локальное
К4	Нет	Нет	Да	Да
К5	Нет	Нет	Нет	Да

Исходя из результатов сравнение можно сделать вывод о том, что лучшим методом для моделирования многофункциональных центров обслуживания является сеть Петри. Конечные автоматы не позволяют отобразить вероятностные события, происходящие в многофункциональных центрах обслуживания. Конечные и вероятностные автоматы не позволят отобразить параллельную работу МФЦ, т. к. придётся вводить большое количество дополнительных состояний, что сильно усложнит модель. Также они не учитывают время. И в отличии от систем массового обслуживания сети Петри концентрируются на локальных событиях и имею больше одного состояния на всю систему, что позволяет получать подробную информацию о каждом элементе в системе. Помимо этого, цветные сети Петри позволяют различать заявки по их типам, чтоб важно, так как разные заявки в МФЦ могут обслуживаться разное время и за них отвечают разные окна.

1.3 Алгоритмы продвижения модельного времени

Динамическая природа дискретно–событийных имитационных моделей требует, требует отслеживания текущего значения имитационного времени по мере функционирования имитационной модели. Также необходим механизм для продвижения имитационного времени от одного значения к другому. В

имитационной модели переменная, переменная текущее значение модельного времени, называется часами модельного времени. Существует два основных подхода к продвижению модельного времени: продвижение времени от события к событию и продвижение времени с постоянным шагом [16].

1.3.1 Алгоритм продвижения времени от события к событию

Характерное свойство систем обработки информации то, что состояния отдельных устройств изменяются в дискретные моменты времени, совпадающие с моментами поступления сообщений в систему, окончания выполнения задания и т.п. При использовании продвижения времени от события к событию часы модельного времени в исходном состоянии устанавливаются в О и определяется время возникновения будущих событий. После этого часы модельного времени переходят на время возникновения ближайшего события, и в этот момент обновляются состояние системы с учётом произошедшего события, а также сведения о времени возникновения будущих событий. Затем часы модельного времени продвигаются ко времени возникновения следующего (нового) ближайшего события, обновляется состояние системы и определяется время будущих событий, и т. д. Процесс продвижения модельного времени от времени возникновения одного события ко времени возникновения другого продолжается до тех пор, пока не будет выполнено какое-либо условие останова, указанное заранее. Поскольку в дискретно–событийной имитационной модели все изменения происходят только во время возникновения событий, периоды бездействия системы просто пропускаются, и часы переводятся со времени возникновения одного события на время возникновения другого. Следует отметить, что длительность интервала продвижения модельного времени от одного события к другому может быть различной [16].

Для работы алгоритма в системе поддерживается список будущих событий, в котором для каждого активного блока, порождающего события заносится, заводится свой элемент. Каждый элемент хранит информацию об объекте и времени следующего события на объекте. Сам же алгоритм выглядит следующим образом:

— в списке находится ближайшее событие;

— найденное событие реализуется;

— ячейка блока, выполнившего событие обновляется, генерируется интервал времени до нового события и прибавляется к текущему времени.

Достоинством этого алгоритма является: отсутствие сложности с подбором шага по времени, достаточно малого, чтоб ничего не пропустить, и достаточно большого, чтоб сократить количество итераций при моделировании.

Основным недостаток этого способа заключается в том, что при большом количестве событий приходится часто выполнять операцию линейного поиска по списку будущих событий, что особенно критично при наличии большого числа активных блоков. Для ускорения поиска можно поддерживать список в виде упорядоченной структуры данных.

1.3.2 Алгоритм продвижения времени с постоянным шагом

Другой подход к продвижению часов модельного времени в дискретно–событийной имитационной модели называется продвижением времени посредством постоянного шага. При таком подходе часы модельного времени двигаются точно на Δt единиц времени/ После каждого обновления часов выполняется проверка с целью определить, произошли ли какие–либо события в течение предыдущего интервала времени Δt. Если на этот интервал запланированы одно или несколько событий, считается, что данные события происходят в конце интервала, после чего состояние системы соответствующим образом обновляется. В ситуациях, когда принято считать, что два или несколько событий происходят в одно и то же время, необходимо применение ряда правил, позволяющих определять, в каком порядке обрабатывать события. Таким образом, продвижение времени посредством постоянного шага имеет два недостатка: возникновение ошибок, связанных с обработкой событий в конце интервала, в течение которого они происходят, а также необходимость решать, какое событие обрабатывать первым, если

события, в действительности происходящие в разное время, рассматриваются как одновременные. Подобного рода проблемы можно частично решить, сделав интервалы Δt менее продолжительными, но тогда возрастает число проверок возникновения событий, что приводит к увеличению времени выполнения задачи. Продвижение времени с помощью постоянного шага не используют в дискретно–событийных имитационных моделях, когда интервалы времени между последовательными событиями могут значительно отличаться по своей продолжительности [17].

Получается основным недостатком этого алгоритма является сложность выбора шага, плохой выбор которого может привести к увеличению затрат машинного времени или получению неадекватных результатов при моделировании.

Основным достоинством является равномерная протяжка времени.

1.3.3 Алгоритм Дельфт

Два приведённых метода являются универсальными алгоритмами протяжки модального времени. Причём для некоторых предметных областей один принцип может работать быстро и без потерь, а другой будет работать неэффективно. Выбор метода необходимо производить исходя из распределения событий по времени. В реальных системах распределение событий, как правило, неоднородно. События, как бы группируются по времени. Образование таких групп связано с наступлением какого–то «значимого» события, которое начинает определённую последовательность действий с соответствующими событиями, имеющими высокую плотность на следующем временном интервале. Такой интервал называется пиковым. А распределение событий квазисинхронным. Примеров может являться приход посетителя в МФЦ, который запускает цепочку событий по его обслуживанию. Для сложных дискретных систем, в которых присутствуют квазиасинхронное распределение событий, был разработан алгоритм с название Дельфт. Особенностью данного метода является автоматическая адаптация к распределению событий. Метод реализуется таким образом, что на пиковых интервалах он приближается к методу с постоянным шагом, а вне пиковых к событийному. В основе лежит использование иерархической структуры циркулярных списков.

Пусть число ячеек N_1 на самом низком уровне равно числу элементов циркулярного списка для пикового интервала, т. е. максимальной длине пиковых интервалов, выраженной в квантах времени. На пиковых интервалах этот список будет действовать в соответствии с пошаговым методом. Для числа ячеек N_2 остальных иерархично расположенных списков, служащих для ускорения шага времени, выбираем равные значения. Таким образом, мы получим желаемую структуру данных, которая состоит из m циркулярных списков индикаторов предсказываемых событий и сопряжённых с ними списков предсказываемых времён T_j Одна ячейка первого циркулярного списка соответствует единице величины приращения времени, тогда как ячейки в других списках соответствуют времени, которое описывается полным списком следующего низшего уровня. Полный временной период, описанный полным поэтапным прохождением до конца структуры списка, должен соответствовать среднему значению моделируемого времени между синхронными событиями [18].

Во время шага моделируемого времени указатели, принадлежащие к часовым структурам, осуществляют полный оборот и таким образом показывают соответствующие интервалы по модулям N_1 и N_2. Ячейки списков от уровня 2 до m содержат числа событий, предсказываемых в соответствующем интервале времени, тогда как ячейки первого циркулярного списка содержат указатели на начало принадлежащих к ним списков событий, если такие имеются [18].

При записи события в циркулярных списках от уровня 2 до m, увеличивается счётчик событий в соответствующей ячейке и событие добавляется в соответствующий список, указательна на который хранится в ячейке 1 уровня. При удалении события соответветственно декрементируется счётчик в списках высокого уровня, и событие удаляется из списка событий ячейки 1 уровня [18].

Шаг моделируемого времени вычисляется предлагаемым алгоритмом с учётом двух ситуаций: в пиковых интервалах, где плотность событий велика, шаг времени определяется элементарным отрезком времени в соответствии с первым циркулярным списком; в интервалах с малой плотностью событий система стремится как можно быстрее «шагнуть» до следующего предсказанного события [18].

Вначале длина шага соответствует ячейкам самого верхнего циркулярного списка, пустые ячейки «перепрыгиваются». Когда обнаруживается непустая ячейка, то осуществляется переход на следующий уровень вниз до тех пор, пока на самом нижнем уровне не обнаружится предсказанный указатель на список событий [18].

Данный алгоритм, адаптируется под распределение событий и тем самым объединяет в себе плюсы методов с постоянным шагом и событийного.

Вывод

Были рассмотри три различных алгоритма протяжки модельного времени, а также их основные достоинства и недостатки. Можно сделать вывод о том, что лучшим из рассмотренных методов является комбинированный Дельфт, так как он подстраивается под распределение событий. Это позволяет ему при большом количестве событий приближаться к пошаговому методу, который хорошо работает в таких случаях, а при малом количестве событий шагать через крупные промежутки времени, как событийный метод.

1.4 Постановка задачи

На основе анализа и сравнения методов моделирования многофункциональных центров обслуживания можно сформулировать следующую цель данной работы: реализовать метод моделирования многофункциональных центров обслуживания на основе сетей Петри и комбинированнного метода продвижения модельного времени. Формально постановка задачи может быть описана с помощью IDEF0–диаграмм нулевого и первого уровня, которые приведены на рисунках 1.6 и 1.7.

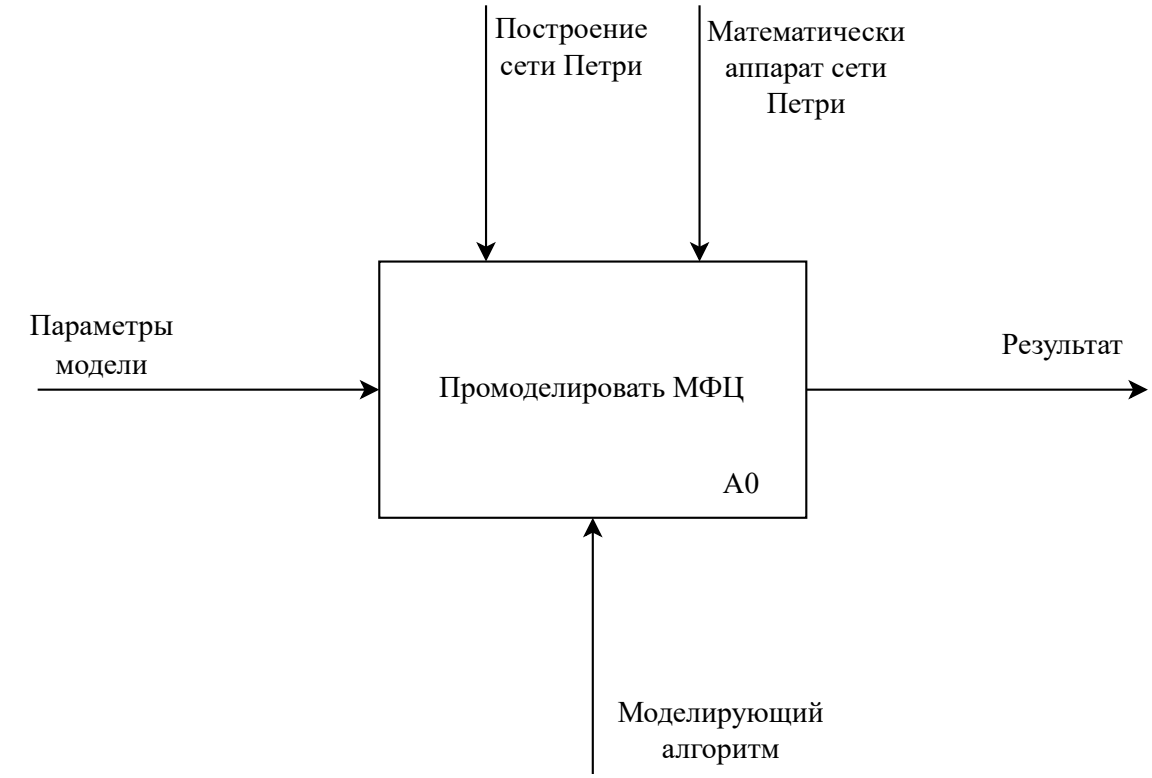

Рисунок 1.6 – Диаграмма постановки задачи нулевого уровня

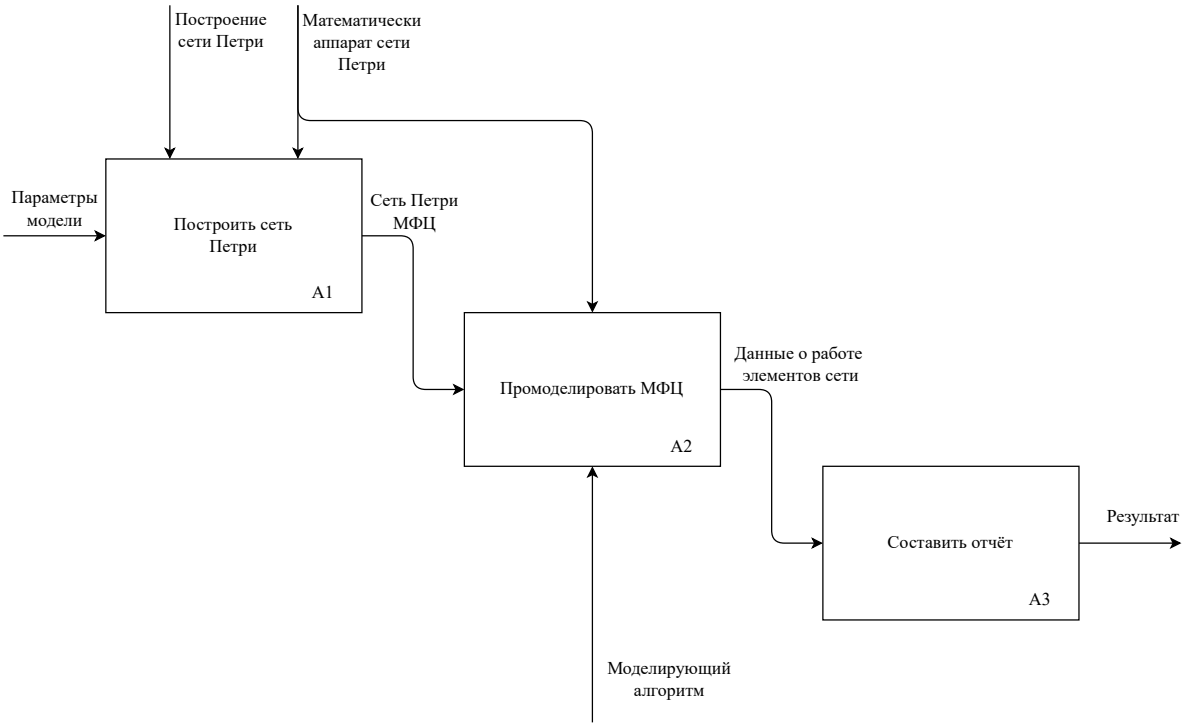

Рисунок 1.7 – Диаграмма постановки задачи первого уровня

Вывод

В данном разделе был проведён анализ предметной области моделирования многофункциональных центров обслуживания. Также были описаны основные формализмы используемые для моделирования многофункциональных центров обслуживания и проведён их сравнительный анализ, в результате которого был сделан вывод, что лучше всех подходят сети Петри. Были рассмотрены основные методы протяжки модельного времени и был выбран комбинированный, как сочетающий основные достоинства базовых алгоритмов. Была описана формальная постановка задачи в виде IDEF0–диаграммы

2 Конструкторский раздел

2.1 Модель многофункциональных центров обслуживания

Многофункциональный центр обслуживания состоит комбинации 3 простейших элементов: генератор, очередь и обслуживающий аппарат, из которых собираются ресепшен и окна обслуживания, а их них весь центр.

2.1.1 Модель обслуживающего аппарата

На рисунке 2.1 представлена модель обслуживающего аппарата.

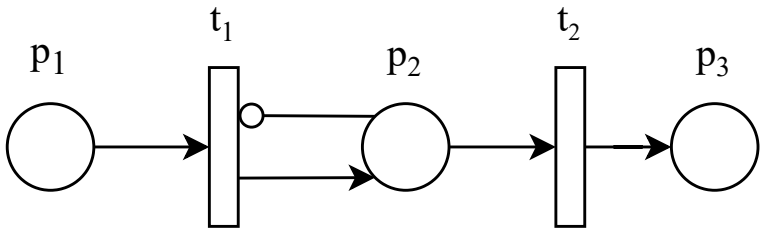

Рисунок 2.1 – Модель обслуживающего аппарата

Описание позиций:

— p_1 — ожидание входа в аппарат;

— p_2 — нахождение в аппарате, обслуживание заявки;

— p_3 — выход из аппарата после обслуживания.

Переход t_1 осуществляет вход в аппарат и активируется, если в p_1 есть хоть одна фишка, а в p_2 не одной, то есть, если есть заявка ожидающая обслуживания и никто не обслуживается, то переход сработает. Проверка отсутствия фишки в p_2 реализована с помощью ингибиторной дуги. В результате срабатывания этого перехода из p_1 в p_2 переносится 1 фишка. Переход t_2 ,срабатывает с задержкой равной времени обслуживания, осуществляет ожидание обработки и выход заявки из аппарата.

2.1.2 Модель очереди

На рисунке 2.2 представлена модель очереди.

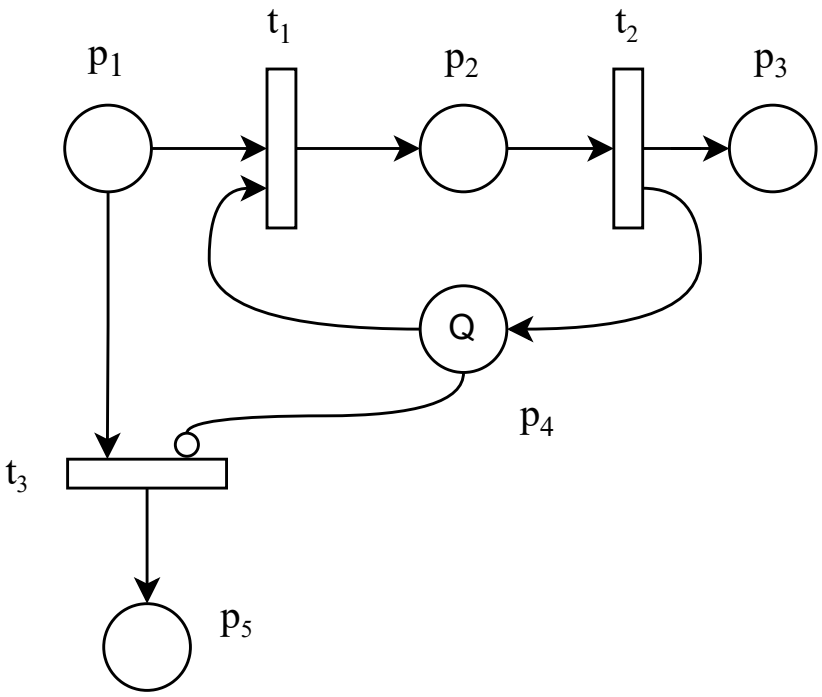

Рисунок 2.2 – Модель очереди

Описание позиций:

— p_1 — ожидание входа в очередь;

— p_2 — нахождение в очереди;

— p_3 — выход из очереди;

— p_4 — ограничитель размера очереди, где Q — количество фишек, максимальный размер очереди, который при входе в p_2 уменьшается, а при выходе увеличивается;

— p_5 — выход из очереди, если она достигла максимального размера.

Описание переходов:

— t_1 — вход в очередь, сработает если в p_1 и p_4 есть фишки, то есть счётчик оставшейся ёмкость очереди больше нуля;

— t_2 — выход из очереди, сработает если p_2 есть фишки, то есть кто-то есть в очереди, и увеличивает счётчик оставшейся ёмкость очереди;

— t_3 — выход из системы, если очередь слишком большая, сработает если не сработал t_1 и нет нет не одной фишки в p_5.

2.1.3 Модель ресепшен

На рисунке 2.3 представлена модель ресепшен.

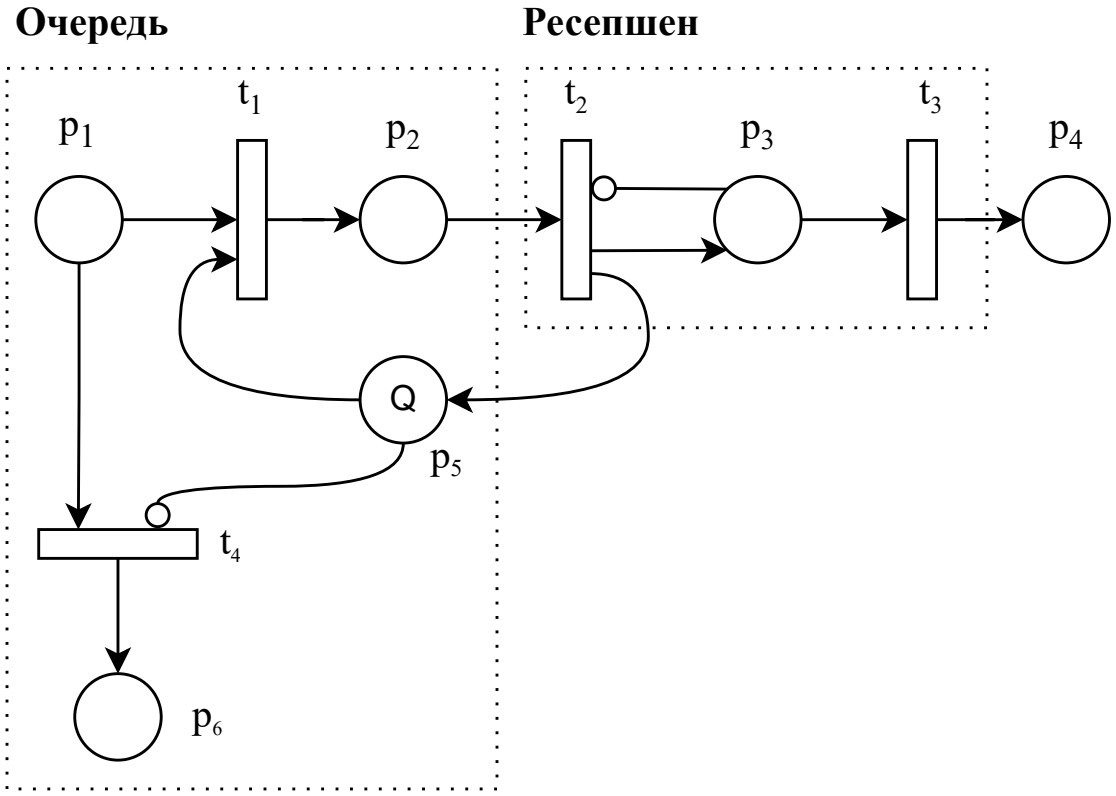

Рисунок 2.3 – Модель ресепшен

Модель ресепшен является объединением очереди и обслуживающего аппарата. Модели объединяются через переход t_2, переход отвечает за выход из очереди и обновление её счётчика и за вход в обслуживающий аппарата. Если есть заявка ожидающая в очереди и никто не обслуживается в аппарате, то переход срабатывает. В позиции p_2 лежат фишки отвечающие за нахождение в очереди, а в p_3 фишка отвечающая за занятие обслуживающего аппарата.

2.1.4 Модель окон обслуживания

На рисунке 2.4 представлена модель окон обслуживания.

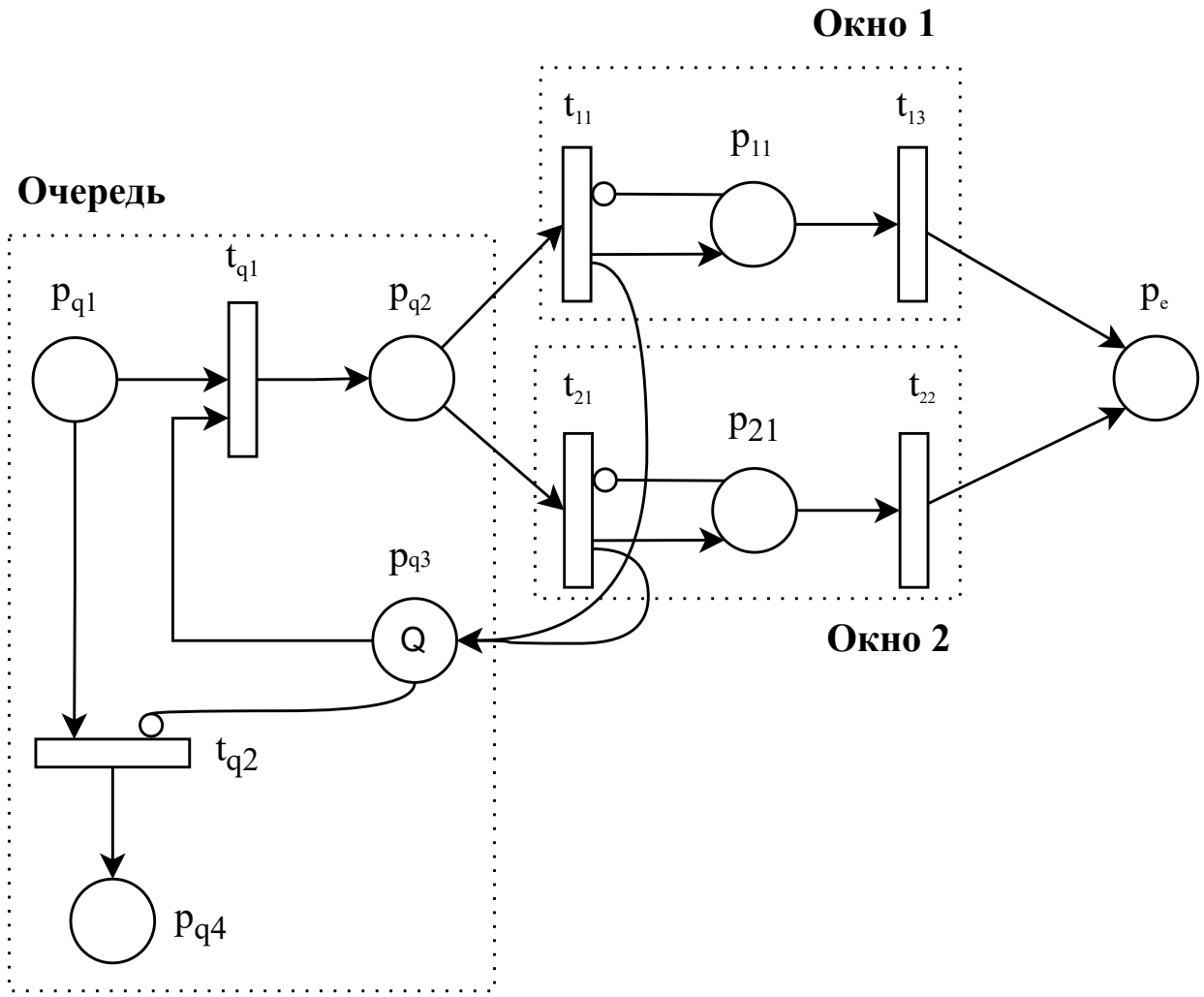

Рисунок 2.4 – Модель окон обслуживания

Модель окон является объединением очереди и нескольких обслуживающих аппаратов аппарата. Модели объединяются через переходы t_{11} и t_{21}, аналогично ресепшен. Оба перехода забирают заявки из очереди, помещают их в обслуживающий аппарат и обновляют счётчик очереди.

2.1.5 Модель многофункционального центра обслуживания

На рисунке 2.5 представлена модель многофункционального центра обслуживания.

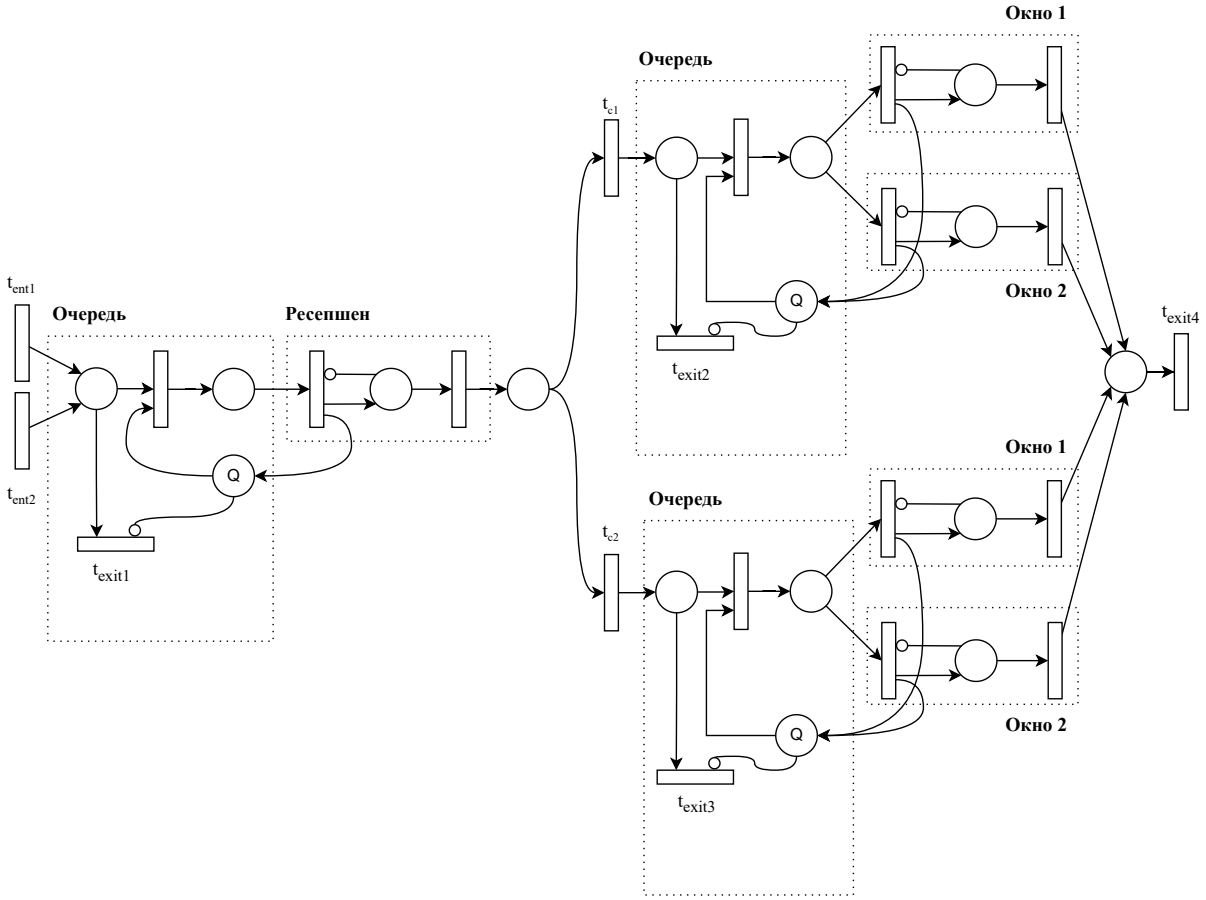

Рисунок 2.5 – Модель многофункционального центра обслуживания

Модель многофункционального центра обслуживания является объединением моделей ресепшена и окон. Соединяются они через переходы t_{c1} и t_{c2}. До этого не упоминалась, что в модели используется цветная сеть Петри, так как не было смысла. В этом примере определён один целочисленный тип (цвет), который описывает тип фишки, каждому типу соответствует своё число. Так как только один цвет, то и все позиции и дуги одного «типа». Также определена свободная переменная x целочисленного типа, используемая в вращениях для переходов t_{ci}. Все переходы, кроме t_{ci}, пропускают фишку с любым значением типа, девствуют как обычные переходы. А переходы t_{ci} срабатывают только при наличии фишки определённого типа или типов,

в связной с ними позиции. Таким образом реализуется разграничение различных видов услуг по времени обработки и окнам. Переходы t_{ent_i} является генераторами фишек различного типа, а переходы t_{exit_i} удаляют фишки, вышедшие из системы.

2.2 Моделирующий алгоритм

В алгоритме определяются 2 переменные M, количество списков, и mlt, шаг списков. В работе алгоритм использует часовую структуру, которая представляет собой список списков, при в списке ниже в mlt раз больше чем текущий список список. Каждая ячейка списка отвечает за определённый промежуток времени. Каждая ячейка соответствует mlt ячейкам уровнем ниже. Во всем списках кроме самого нижнего, где ячейка соответствует элементарному шагу в системе, лежит количество событий, которое должно произойти в соответствующий ячейке промежуток времени. Нижний же список содержит указатели непосредственно на списки событий, которые должны произойти. Алгоритм последовательно перебирает, ячейка пока не находит ненулевою, спускается до нулевого уровня, выполняет события и возвращается вверх по спискам для продолжения поиска событий. Это позволяет алгоритму проматывать большие пустые интервалы времени , как событийный метод, и в тоже время пр наличии событий работать как метод с постоянным шагом. Схема алгоритма изображена на рисунках 2.6– 2.10.

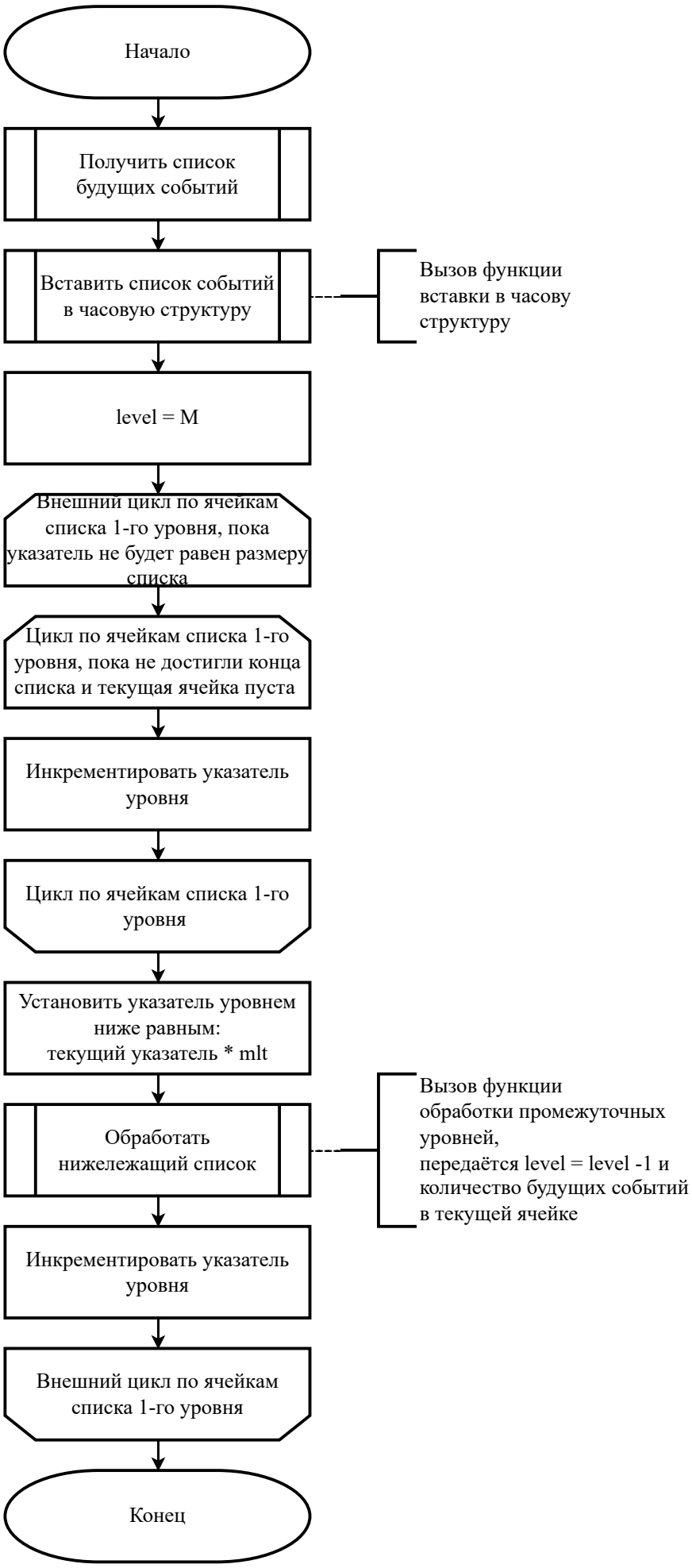

Рисунок 2.6 – Схема функции обработки верхнего списка

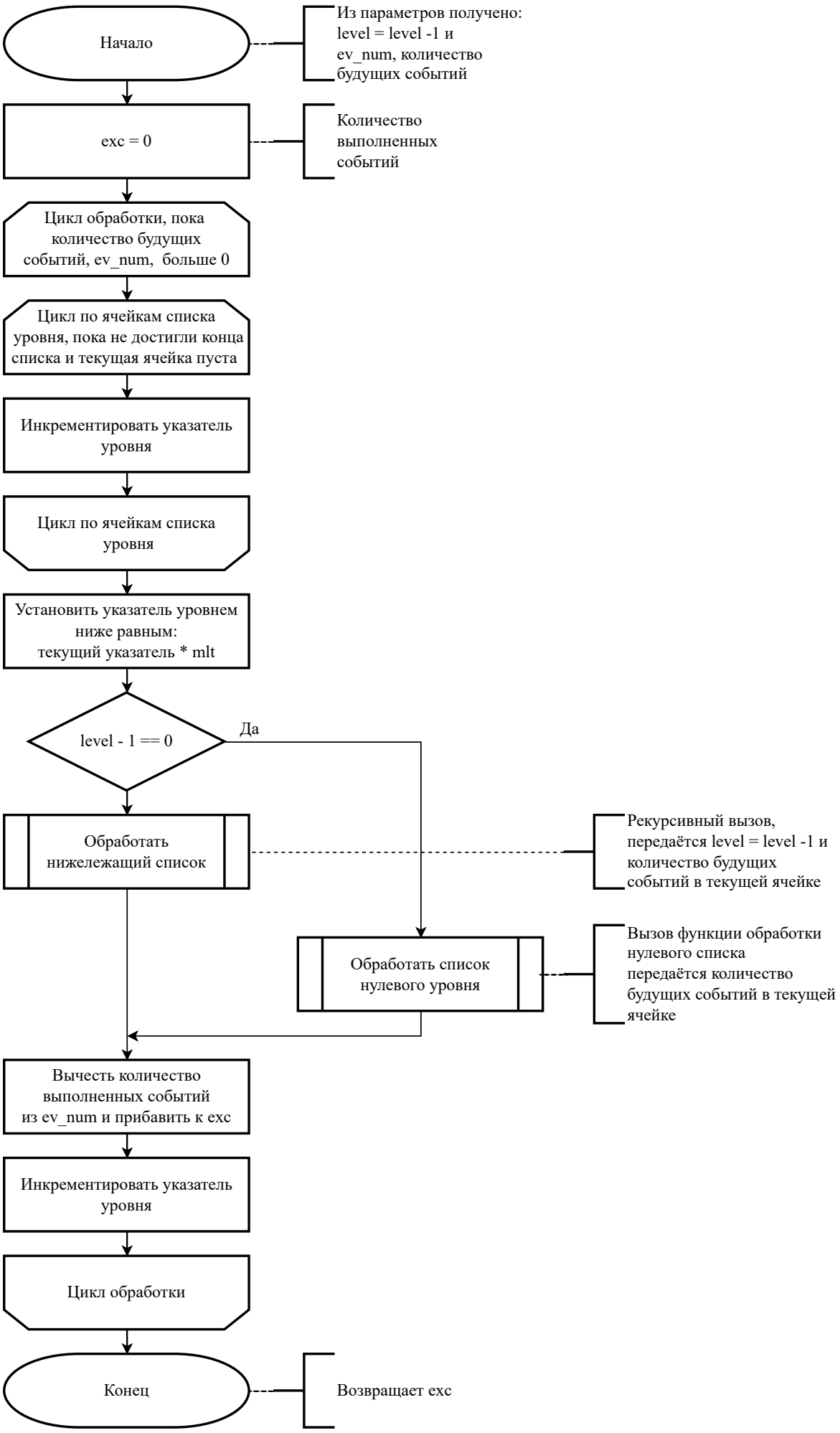

Рисунок 2.7 – Схема функции обработки промежуточных списков

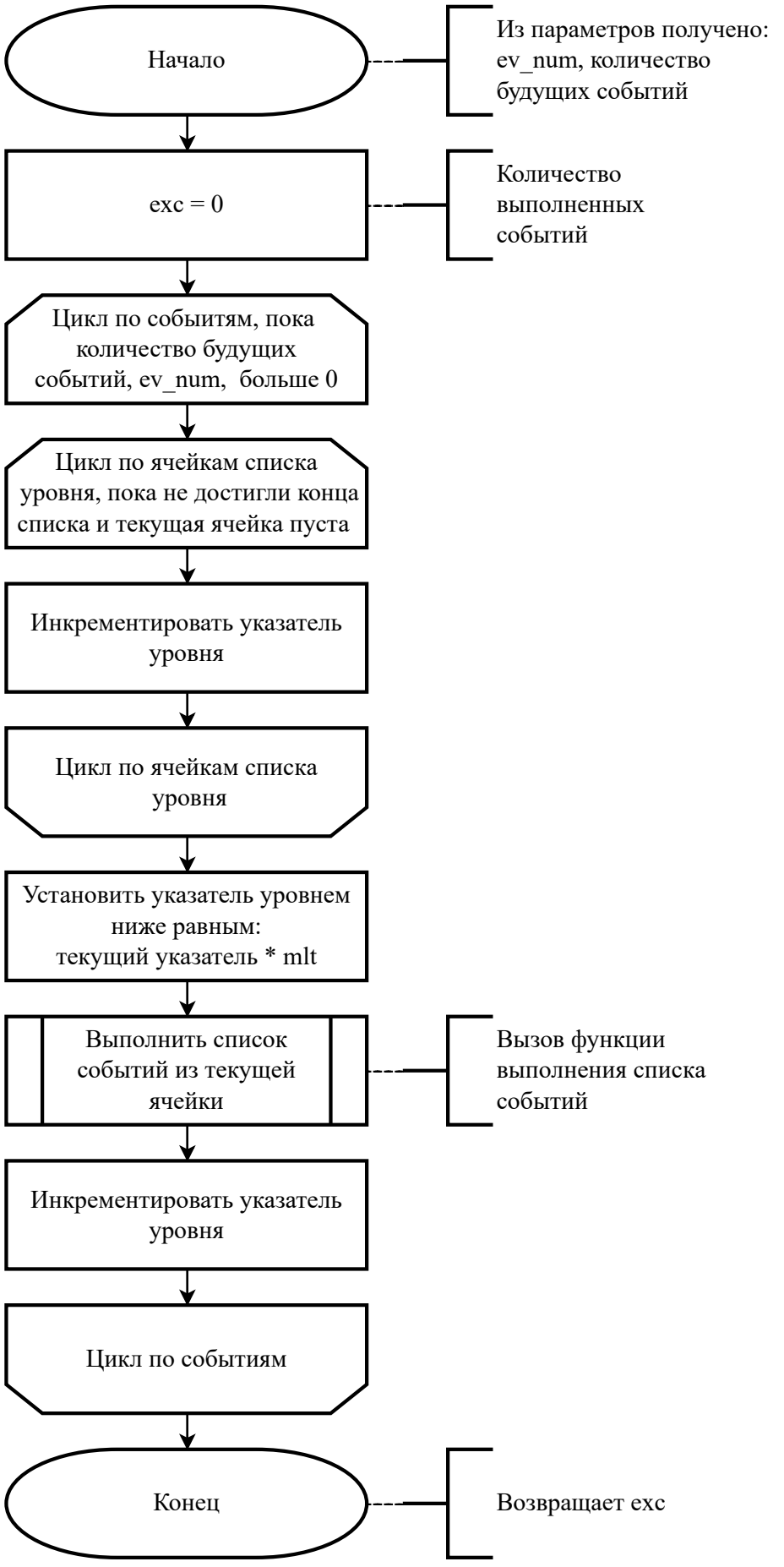

Рисунок 2.8 – Схема функции обработки нижнего списка

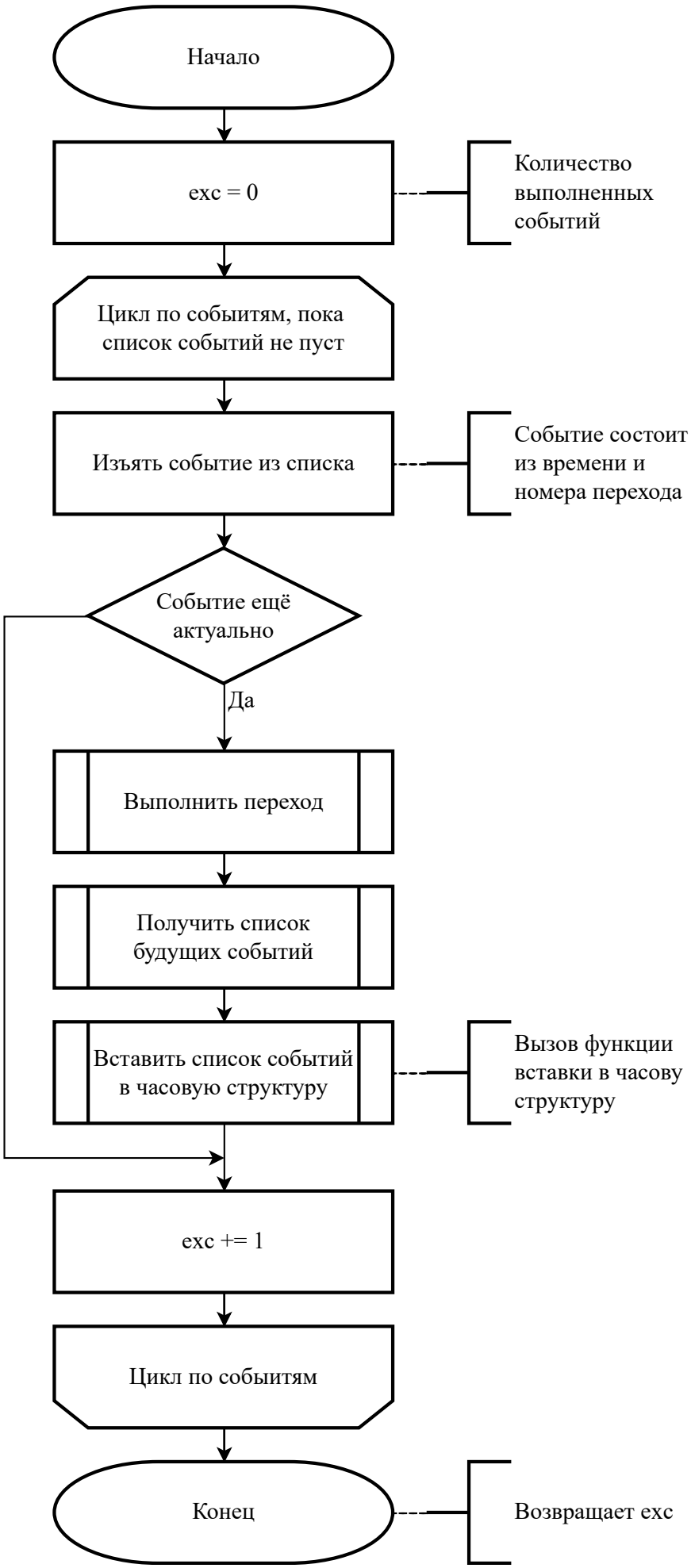

Рисунок 2.9 – Схема функции обработки ячейки нижнего списка

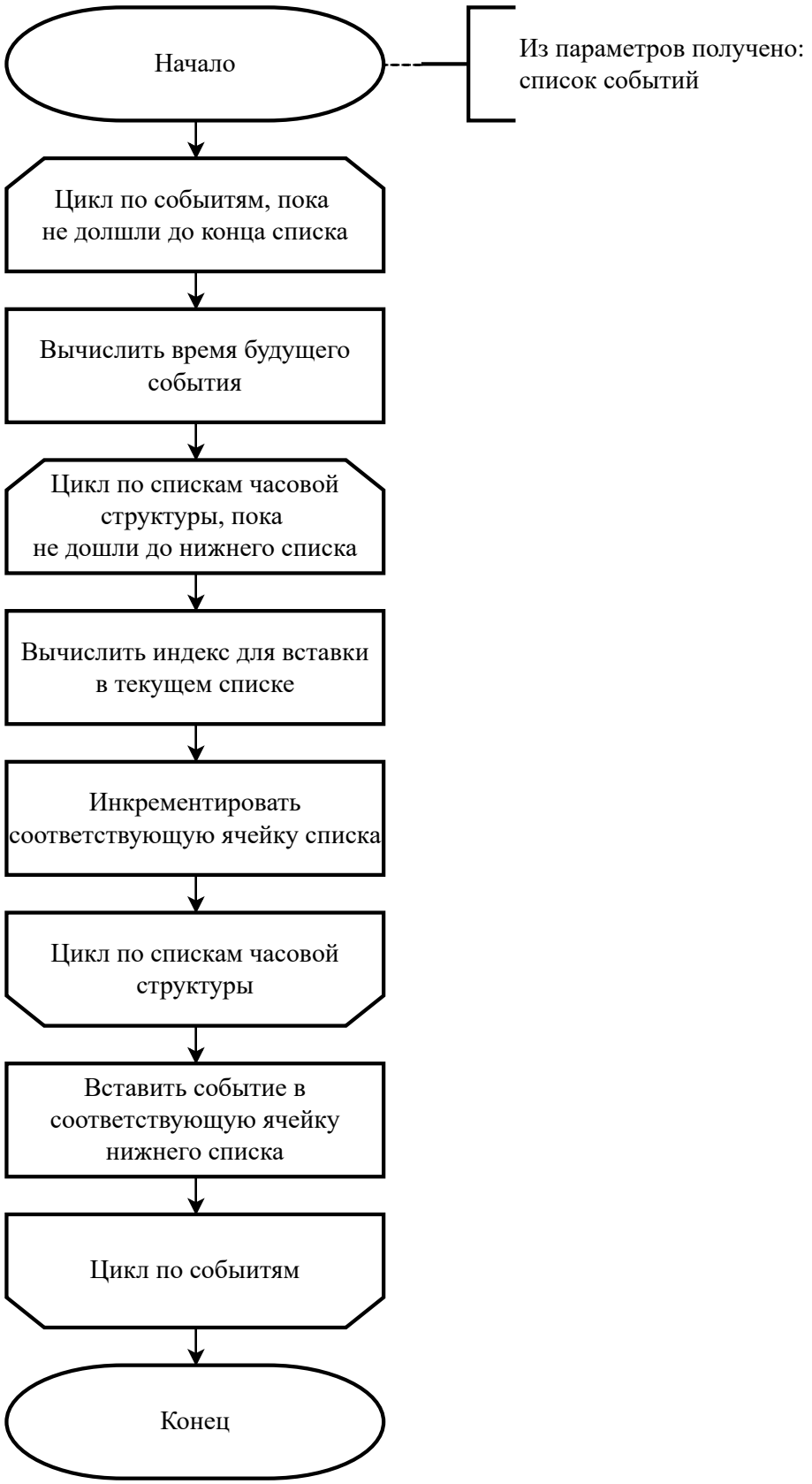

Рисунок 2.10 – Схема функции вставки события

Вывод

В данном разделе были представлены модели примитивов, из которых строятся модели элементов многофункциональных центров. Также были описаны как сами элементы центров обслуживания, так и полностью собранная модель многофункциональных центров обслуживания. Были приведены схемы алгоритмов функционирования сетей Петри и моделирующего алгоритма.

3 Технологический раздел

3.1 Выбор программных средств реализации

Для реализации программного обеспечения был выбран язык программирования C++. Данный язык был выбран по следующим причинам:

— в нём используется объектно–ориентированная парадигма;

— наличие стандартной библиотеки с готовыми контейнерами и алгоритмами над ними;

— наличие строгой типизации, что упрощает процесс отладки;

— наличие необходимых библиотек для реализации графического интерфейса.

Для реализации графического интерфейса был выбран фреймворк Qt, по следующим причинам:

— поддержка C++;

— наличие ПО для создания графических окон;

— широкая библиотека элементов для графического интерфейса и возможность их модернизации.

В качестве среды разработки был выбран Clion, по следующим причинам:

— встроенная поддержка средств отладки, анализа кода и сборки;

— поддержка для Qt;

— наличии студенческой лицензии.

3.2 Описание пользовательского интерфейса интерфейса

Реализованное ПО получает на вход характеристики многофункционального центра обслуживания, а на выходе выдаёт данные о работе элементов системы. На вход программе поступают информация о группах посетителей, о ресепшен и о группах окон обслуживания. Также пользователь вводит количество групп пользователей и окон и время моделирования.

Для **групп посетителей** вводятся следующие данные:

— закон распределения интервала прихода посетителя;

— разброс интервала прихода;

— закон распределения интервала обработки;

— разброс интервала обработки.

Для **ресепшен** вводятся следующие данные:

— закон распределения интервала обработки;

— разброс интервала обработки;

— максимальный размер очереди, начиная с которого посетители будут покидать систему без обслуживания.

Для **групп окон** вводятся следующие данные:

— количество окон в группе;

— максимальный размер очереди, начиная с которого посетители будут покидать систему без обслуживания;

— группы посетителей, которые обрабатывает данные окна.

После моделирования собираются данные по интересующим переходам и позициям, они обрабатываются и выводятся на экран. Выводится информация о работе все системы, а также каждой очереди, ресепшена и групп окон.

Для **системы** выводится:

— количество пришедших клиентов;

— процент успешно обслуженных клиентов;

— процент клиентов, ушедших из системы без обслуживания, из–за большой очереди.

Сумма успешно обслуженных клиентов и ушедших может быть не равна количеству пришедших, так как на момент конца моделирования в системе остались ещё не вышедшие заявки.

Для **очередей** выводится:

— название;

— максимальный размер;

— количество попыток входа (удачные и неудачные);

— количество входов в пустую очередь;

— средний размер очереди;

— средний время пребывания клиента в очереди;

— количество клиентов, которые не смогли встать в очередь из–за её большого размера.

Для **обработчиков** выводится:

— название;

— количество успешно обработнных клиентов;

— среднее относительное время работы окон;

— среднее время работы окон.

3.3 Пример работы программы

На рисунке 3.1 подставлен интерфейс разработанной программы, в котором можно ввести параметры центра обслуживания описанные выше.

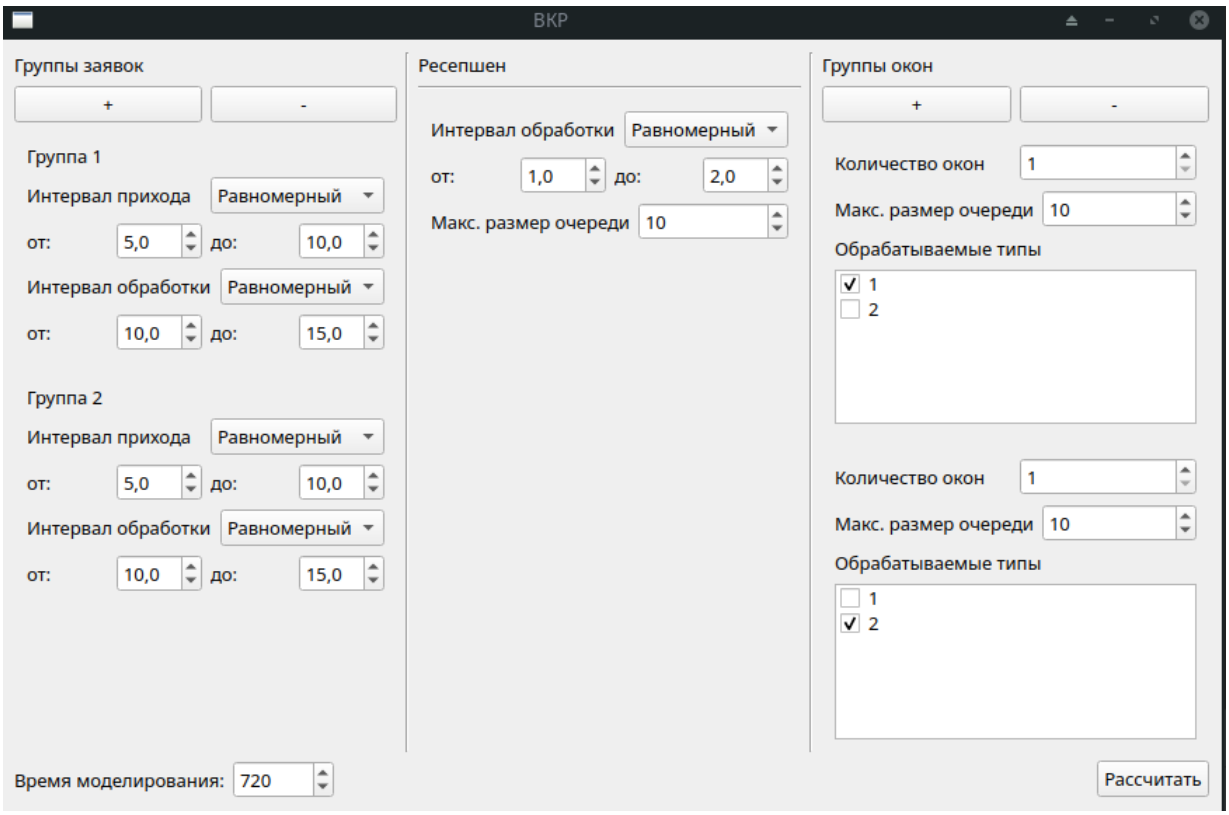

Рисунок 3.1 – Ввод параметров многофункционального центра обслуживания

На рисунке 3.2 подставлен результат моделирования, смысл значений которого были описаны выше, для параметров изображённых на рисунке 3.1.

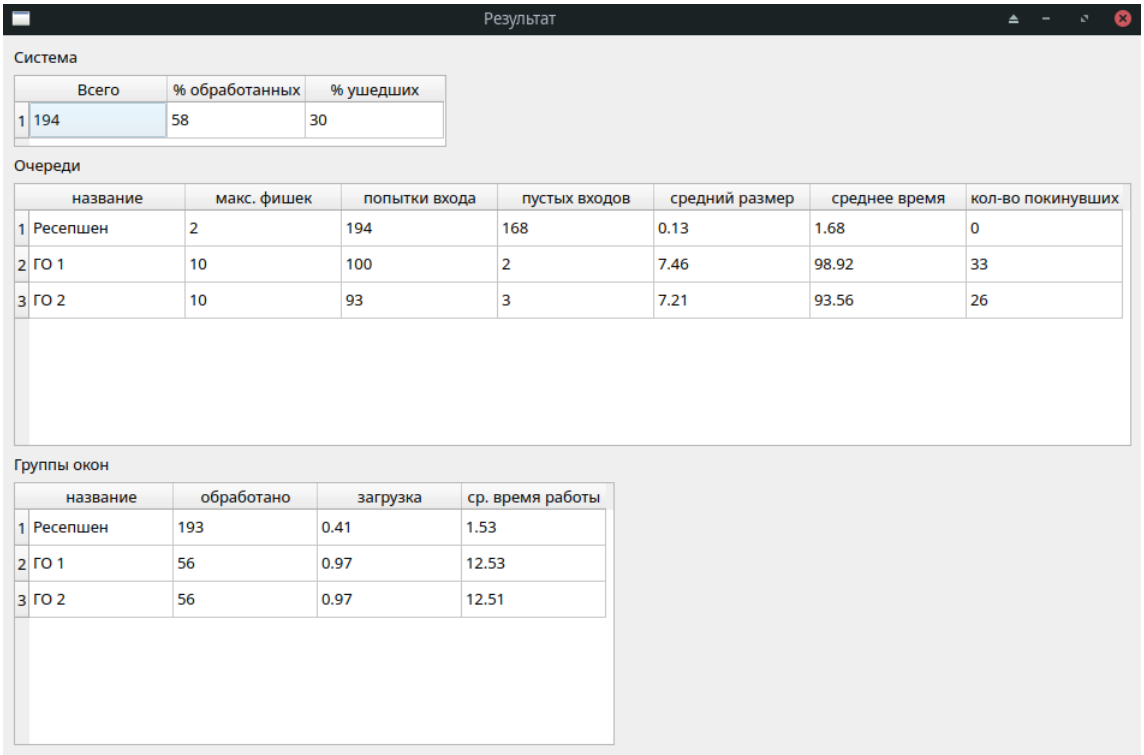

Рисунок 3.2 – Результаты моделирования

3.4 Реализация программного обеспечения

Разработанное ПО состоит из следующих логических частей:

— сеть Петри: построение, функционирование и сбор статистических данных;

— моделирующий алгоритм: управление сетью Петри;

— графический интерфейс.

3.4.1 Реализация сети Петри

В листингах 3.1–3.3 приведена реализация основного функционала сети Петри, используемого в моделирующем алгоритме.

В листинге 3.1, представлен код функции поиска активных переходов. Перебираются все переходы, кроме уже запланированных на выполнение, и проверяются. В результирующий массив попадает индекс перехода и длительность срабатывания перехода.

Листинг 3.1 – Функция поиска активного перехода

```
vector<PetriEvent> PetriNet::find_fired_t_init() {
    vector<PetriEvent> res;

    for (int i = 0; i < t_num; ++i) {
        if (is_wait[i]) continue;

        auto fire_res = is_t_fire(i);
        if (fire_res.first) {
            if (fire_res.second > 0) is_wait[i] = true;
            res.push_back({i, fire_res.second});
        }
    }

    return res;
}
```

В листинге 3.2, представлен код функций проверки активности перехода. В зависимости от функционала перехода, выбирается функция проверки. Для переходов, которые пропускают только определённый тип фишек, проверятся наличие таковых в позиции. Для переходов, которые отражают работу окна, проверяется наличие фишек в соответствующей позиции и генерируется длительность срабатывания перехода, в зависимости от типа фишки. Для обычных переходов, проверяются все связанные дугами, в том числе ингибиторными, позиции на наличие необходимого числа фишек.

Листинг 3.2 – Функции проверки автивности перехода

```
pair<bool, double> PetriNet::is_t_fire(int t_i) {
    if (selector_t.contains(t_i)) return check_selector_t(t_i);
    else if (win_poc.contains(t_i)) return check_win_proc_t(t_i);
    return check_usual_t(t_i);
}

pair<bool, double> PetriNet::check_selector_t(int t_i) {
    auto eff = cpn_t_effect[t_i];
    auto tokens = m[eff.pop_p];
    auto need_tokens = selector_t[t_i];

    for (auto &it: tokens) {
        if (need_tokens.contains(it)) return {true,
           timing[t_i]->gen()};
    }

    return {false, 0};
}

pair<bool, double> PetriNet::check_win_proc_t(int t_i) {
    auto eff = cpn_t_effect[t_i];
    auto tokens = m[eff.pop_p];

    if (tokens.empty()) return {false, 0};

    auto type = tokens.front();
    return {true, type_proc_distro[type]->gen()};
}

pair<bool, double> PetriNet::check_usual_t(int t_i) {
```

```
30      for (auto const &it: t_check[t_i])
31          if (m[it.p_index].size() < it.min_num ||
               !m[it.p_index].empty() && it.is_ing)
32              return {false, 0};
33
34      return {true, timing[t_i]->gen()};
35  }
```

В листинге 3.3, представлен код функций срабатывания перехода. В зависимости от функционала перехода, выбирается функция срабатывания. Для всех переходов снимается флаг планирования, чтоб их проверяли на активность. Для генераторов, добавляются фишки соответствующего типа в позицию. Для переходов, которые пропускают только определённый тип фишек, происходит выбор фишки нужного типа и перенос её в выходную позицию. Для обычных переходов, переносится верхняя фишка из очереди.

Листинг 3.3 – Функции срабатывания перехода

```
1  void PetriNet::fire_t(PetriEvent event) {
2      t_stats[event.t_i].fire_num++;
3
4      int t_i = event.t_i;
5      is_wait[t_i] = false;
6      if (gen_t.contains(t_i)) {
7          process_gen_t(event);
8      } else if (selector_t.contains(t_i)) {
9          process_selector_t(event);
10     } else {
11         process_usual_t(event);
12     }
13 }
14
15 void PetriNet::process_gen_t(PetriEvent event) {
16     int t_i = event.t_i;
17     auto eff = cpn_t_effect[t_i];
18
19     push_p_stats(eff.push_p);
20     m[eff.push_p].push_back(gen_t[t_i]);
21     logs.push_back({t_i, eff.pop_p, eff.push_p, gen_t[t_i],
          event.gen_time, event.sys_time});
```

```
22 }
23
24 void PetriNet::process_selector_t(PetriEvent event) {
25     int t_i = event.t_i;
26     auto eff = cpn_t_effect[t_i];
27     auto &tokens = m[eff.pop_p];
28     auto need_tokens = selector_t[t_i];
29
30     int val = -1;
31     for (auto it = begin(tokens); it != end(tokens); ++it) {
32         if (need_tokens.contains(*it)) {
33             val = *it;
34             tokens.erase(it);
35             break;
36         }
37     }
38
39     if (eff.add_p >= 0) {
40         if (eff.add_val > 0) {
41             push_p_stats(eff.add_p);
42             m[eff.add_p].push_back(-1);
43             logs.push_back({t_i, -1, eff.add_p, -1, event.gen_time,
                   event.sys_time});
44         } else {
45             m[eff.add_p].pop_front();
46             logs.push_back({t_i, eff.add_p, -1, -1, event.gen_time,
                   event.sys_time});
47         }
48     }
49
50     int to = -1;
51     if (eff.push_p >= 0) {
52         push_p_stats(eff.push_p);
53         m[eff.push_p].push_back(val);
54         to = eff.push_p;
55     }
56     logs.push_back({t_i, eff.pop_p, to, val, event.gen_time,
          event.sys_time});
57 }
58
59 void PetriNet::process_usual_t(PetriEvent event) {
```

```
60      int t_i = event.t_i;
61      auto eff = cpn_t_effect[t_i];
62
63      if (eff.add_p >= 0) {
64          if (eff.add_val > 0) {
65              push_p_stats(eff.add_p);
66              m[eff.add_p].push_back(-1);
67              logs.push_back({t_i, -1, eff.add_p, -1, event.gen_time,
                    event.sys_time});
68          } else {
69              m[eff.add_p].pop_front();
70              logs.push_back({t_i, eff.add_p, -1, -1, event.gen_time,
                    event.sys_time});
71          }
72      }
73
74      int val = m[eff.pop_p].front();
75      m[eff.pop_p].pop_front();
76      int to = -1;
77      if (eff.push_p >= 0) {
78          push_p_stats(eff.push_p);
79          m[eff.push_p].push_back(val);
80          to = eff.push_p;
81      }
82      logs.push_back({t_i, eff.pop_p, to, val, event.gen_time,
            event.sys_time});
83  }
```

3.4.2 Реализация моделирующего алгоритма

В листинге 3.4 приведён класс представляющий моделирующий алгоритм.

Листинг 3.4 – Класс моделирующего алгоритма

```
1  struct DelftParam {
2      int step;
3      int mult;
4      int m_time;
5  };
```

```
6
7  class Delft {
8      private:
9      DelftParam param;
10
11     vector<queue<PetriEvent>> low_list;
12     int low_pointer;
13     int low_period;
14
15     vector<vector<int>> top_lists;
16     vector<int> level_pointer;
17     vector<int> level_period;
18
19     shared_ptr<PetriNet> net;
20     public:
21     Delft(DelftParam param, shared_ptr<PetriNet> net);
22
23     void run();
24
25     private:
26     int execute(int level, int &event_num);
27
28     int execute_zero_lvl(int &event_num);
29
30     int execute_events();
31
32     void insert_events(const vector<PetriEvent> &events, double
           now);
33 };
```

В листинге 3.5 приведён код функции, которая обрабатывает самый верхний список часовой структуры. Сначала вставляются события, готовые к выполнению с самого начала. Далее в цикле проходятся по ячейкам списка, пока не найдут не пустую, значение ячейки говорит о количестве событий запланированных в этом промежутке времени. При наличии событий, вызывается функция обработки нижележащего списка и инкрементируется указатель списка.

Листинг 3.5 – Функция обработки верхнего списка

```
1  void Delft::run() {
2      insert_events(net->find_fired_t_init(), 0);
3
4      int level = top_lists.size() - 1;
5      while (true) {
6          while (level_pointer[level] < top_lists[level].size() &&
               top_lists[level][level_pointer[level]] == 0)
7          level_pointer[level]++;
8          if (level_pointer[level] == top_lists[level].size() &&
               top_lists[level][level_pointer[level] - 1] == 0)
9          break;
10
11         if (level - 1 < 0) {
12             low_pointer = level_pointer[level] * param.mult;
13             execute_zero_lvl(top_lists[level][level_pointer[level]]);
14         } else {
15             level_pointer[level - 1] = level_pointer[level] *
                   param.mult;
16             execute(level - 1,
                   top_lists[level][level_pointer[level]]);
17         }
18
19         level_pointer[level]++;
20     }
21 }
```

В листинге 3.6 приведён код функции, которая обрабатывает промежуточные списки часовой структуры. Работает аналогично функции обработки верхнего уровня, только дополнительно декрементирует счётчик событий ячейки уровнем выше и возвращает количество обработанных событий.

Листинг 3.6 – Функция обработки промежуточных списков

```
1  int Delft::execute(int level, int &event_num) {
2      int event_done = 0;
3
4      while (event_num > 0) {
5          while (level_pointer[level] < top_lists[level].size() &&
               top_lists[level][level_pointer[level]] == 0)
6          level_pointer[level]++;
7
```

```
8           int exc = 0;
9           if (level - 1 < 0) {
10              low_pointer = level_pointer[level] * param.mult;
11              exc =
                    execute_zero_lvl(top_lists[level][level_pointer[level]]);
12          } else {
13              level_pointer[level - 1] = level_pointer[level] *
                    param.mult;
14              exc = execute(level - 1,
                    top_lists[level][level_pointer[level]]);
15          }
16
17          event_num -= exc;
18          event_done += exc;
19
20          level_pointer[level]++;
21      }
22
23      return event_done;
24  }
```

В листинге 3.7 приведён код функции, которая обрабатывает список нулевого уровня часовой структуры. Работает аналогично функции обработки промежуточных уровней уровня, только вместо обработки нижележащих уровней начинает выполнение событий.

Листинг 3.7 – Функция обработки нижнего списка

```
1  int Delft::execute_zero_lvl(int &event_num) {
2      int event_done = 0;
3
4      while (event_num > 0) {
5          while (low_pointer < low_list.size() &&
                   low_list[low_pointer].empty())
6          low_pointer++;
7
8          int exc = execute_events();
9
10         event_num -= exc;
11         event_done += exc;
12
```

```
13          low_pointer++;
14      }
15
16      return event_done;
17  }
```

В листинге 3.8 приведён код функции, выполняет события из ячейки списка нулевого уровня. Пока список событий не пуст выполняется все события лежащие в нём, а именно активируются переходы. Перед выполнением дополнительно проверяется возможность активации перехода, из–за конфликтов в сети. После выполнения снова происходит поиск активных переходов и вставка их в часовую структуру, при это новое событие может попасть прямо в обрабатываемую очередь.

Листинг 3.8 – Функция выполнения событий

```
1  int Delft::execute_events() {
2      int event_done = 0;
3      auto &q = low_list[low_pointer];
4
5      while (!q.empty()) {
6          event_done++;
7
8          auto event = q.front();
9          q.pop();
10         auto check_res = net->is_t_fire(event.t_i);
11         if (!check_res.first) continue;
12
13         net->fire_t(event);
14         insert_events(net->find_fired_t_init(), event.sys_time);
15     }
16
17     return event_done;
18 }
```

В листинге 3.9 приведён код функции вставки событий в часовую структуру. Для события последовательно вычисляются индексы в верхних списках и инкрементируется счётчики в соответствующих ячейках. Для ячейки нижнего уровня происходит вставка в ей список событий. Если событие должно

выполниться моментально, то просто происходит вставка по текущим индексам.

Листинг 3.9 – Функция выполнения событий

```
1  void Delft::insert_events(const vector<PetriEvent> &events, double
       now) {
2      for (auto it: events) {
3          if (it.gen_time == 0) {
4              it.sys_time = now;
5              for (int i = 0; i < top_lists.size(); ++i)
                   top_lists[i][level_pointer[i]]++;
6              low_list[low_pointer].push(it);
7          } else {
8              double time = now + it.gen_time;
9              if (time > param.m_time)continue;
10
11             it.sys_time = time;
12             time -= EPS;
13
14             for (int i = 0; i < top_lists.size(); ++i) {
15                 int index = int(time / level_period[i]);
16                 top_lists[i][index]++;
17             }
18
19             low_list[int(time / low_period)].push(it);
20         }
21     }
22 }
```

Вывод

В данном разделе был приведён выбор средств программной реализации. Описан интерфейс программы, а также входные и выходные данные. Представлена и описана основная часть реализации сетей Петри и моделирующего алгоритма.

4 Исследовательский раздел

4.1 Исследование времени работы ПО

В качестве модели взят многофункциональный центр с тремя генераторами и тремя группами окон по три окна каждая. Каждое окно обрабатывает только один тип заявок. Временные характеристики типов заявок идентичны. В моделирующем алгоритме помимо времени моделирования ещё есть параметры элементарного шага и множитель уровней. На рисунка 4.1– 4.3 представлены результаты замеров времени работы алгоритма в зависимости от загрузки, шага и множителя уровней. На каждом изображении нарисованы графики зависимости времени работы от загрузки, для различных шагов. Шаг варьируются в интервале от 1 до 5, так как минимальный интервал времени в системе равен 5. Если поставить больше, то события могут выполнятся в неверном порядке, так как внутри ячейки нижнего уровня они не сортируются. Для каждого рисунка используется разный множитель уровней при котором в часовой структуре 11, 5 и 3 уровня соответственно. Время моделирования 2000. Замеры проводились по 10 раз и бралось среднее.

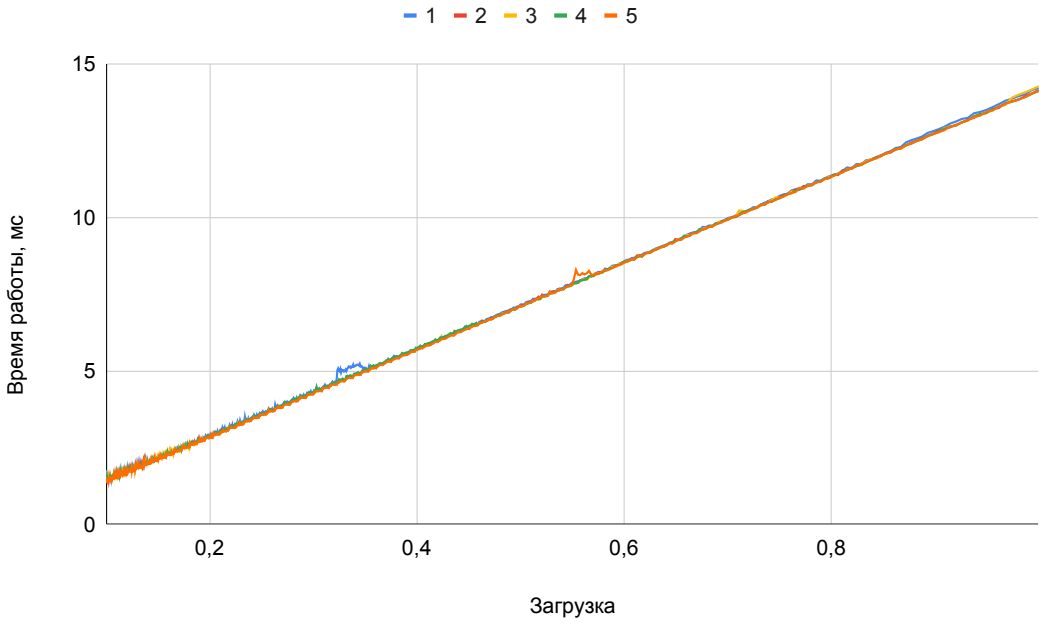

Рисунок 4.1 – Зависимость времени работы от загрузки при множителе равным 2

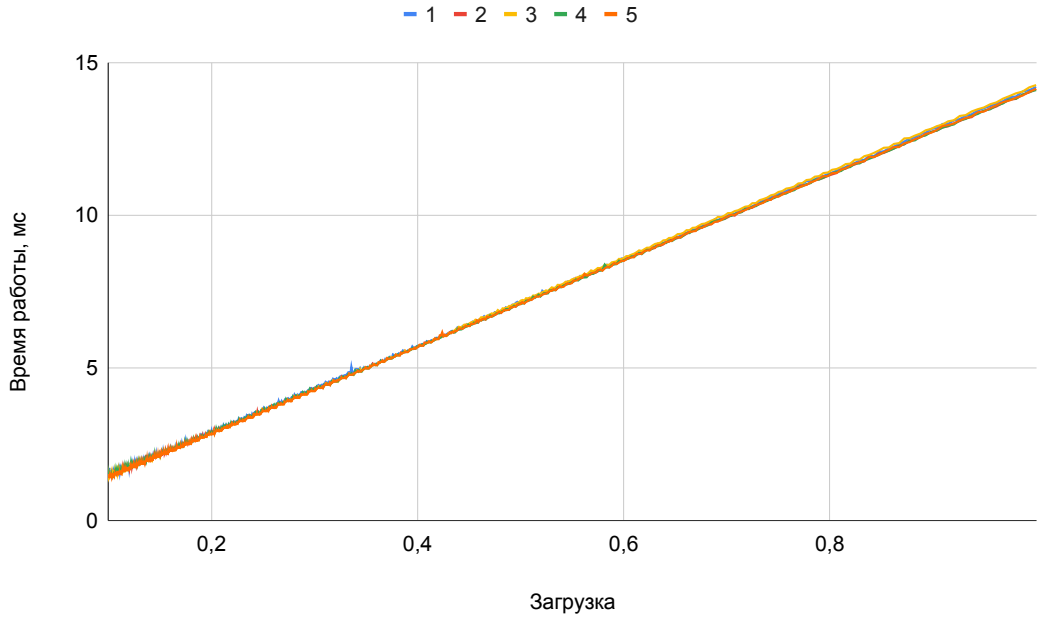

Рисунок 4.2 – Зависимость времени работы от загрузки при множителе равным 6

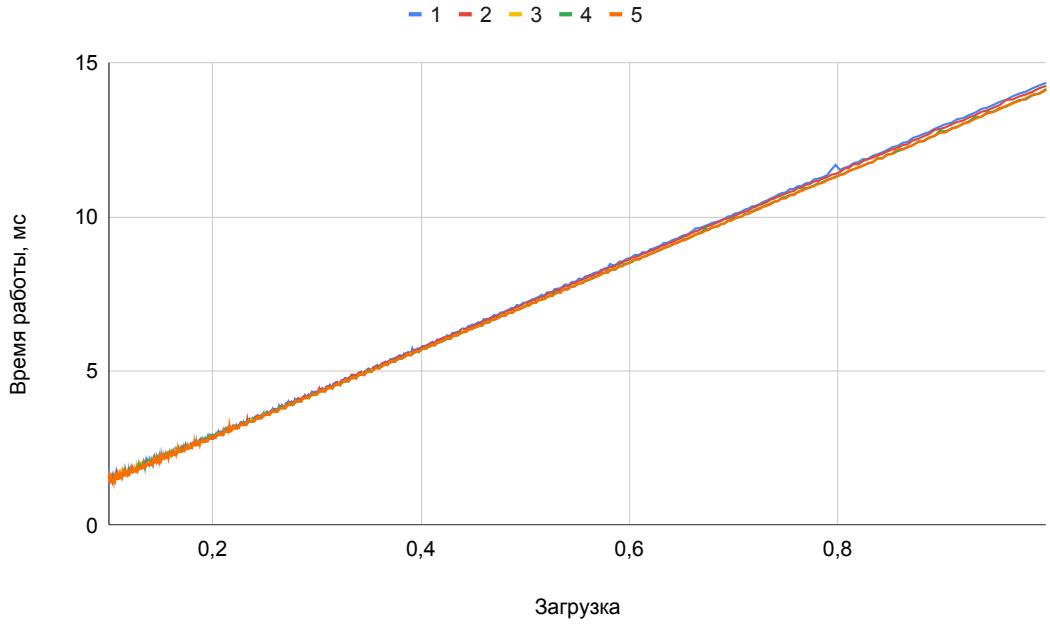

Рисунок 4.3 – Зависимость времени работы от загрузки при множителе равным 10

Из результатов видно, что множитель уровней не оказывать почти никакого влияния в рассмотренном случае. Также видно, что программа с шагом 5 работает быстрее всего, приморено на 1% быстрее шага 2, самый медленный. Можно сделать вывод, что не стоит брать шаг равным 2, а также о

линейном виде зависимости времени работы от загрузки системы.

На рисунке 4.4 изображена зависимость времени работы от времени моделирования для различных загрузок. Модель аналогична прошлой.

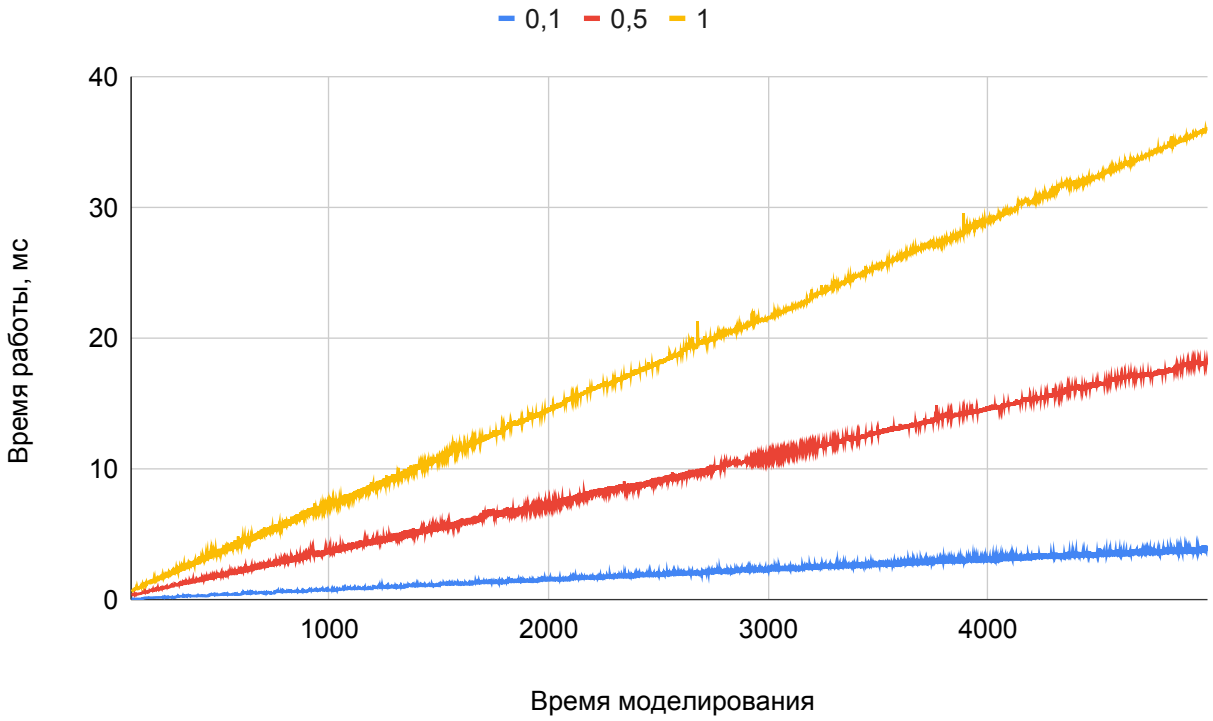

Рисунок 4.4 – Зависимость времени работы от времени моделирования

Из результатов видно что зависимость имеет линейный характер. Также чем больше нагрузку тем быстрее растёт время.

4.2 Сравнений с GPSS World

Используется модель такая же как и ранее. В листинге 4.1 приведено описание аналогичной модели используемой в GPSS World с загрузкой 0.5 и временем моделирования 2000.

Листинг 4.1 – Модель GPSS World

```
1 SIMULATE
2 wg_1      STORAGE  3
3 wg_2      STORAGE  3
4 wg_3      STORAGE  3
5
```

```
6  GENERATE 30,0.1
7  ASSIGN type,1
8  TRANSFER ,recep
9
10 GENERATE 30,0.1
11 ASSIGN type,2
12 TRANSFER ,recep
13
14 GENERATE 30,0.1
15 ASSIGN type,3
16 TRANSFER ,recep
17
18
19 recep   TEST L q$reception_q,10,out_q
20 QUEUE reception_q
21 SEIZE reception
22 DEPART reception_q
23 ADVANCE 5,0.1
24 RELEASE reception
25
26 TEST L P$type,3,wg_3_p
27 TEST L P$type,2,wg_2_p
28
29
30 wg_1_p  TEST L q$wg_1_q,10,out_wg_1
31 QUEUE wg_1_q
32 ENTER wg_1
33 DEPART wg_1_q
34 ADVANCE 45,0.1
35 LEAVE wg_1
36 TRANSFER ,out
37
38 wg_2_p  TEST L q$wg_2_q,10,out_wg_2
39 QUEUE wg_2_q
40 ENTER wg_2
41 DEPART wg_2_q
42 ADVANCE 45,0.1
43 LEAVE wg_2
44 TRANSFER ,out
45
46 wg_3_p  TEST L q$wg_3_q,10,out_wg_3
```

```
47 QUEUE wg_3_q
48 ENTER wg_3
49 DEPART wg_3_q
50 ADVANCE 45,0.1
51 LEAVE wg_3
52 TRANSFER ,out
53
54
55 out TERMINATE
56 out_q    TERMINATE
57 out_wg_1     TERMINATE
58 out_wg_2     TERMINATE
59 out_wg_3     TERMINATE
60
61
62 GENERATE 2000
63 TERMINATE 1
64 START 1
```

На рисунке 4.5 изображён график зависимости времени работы от загрузки для реализованного ПО и GPSS World.

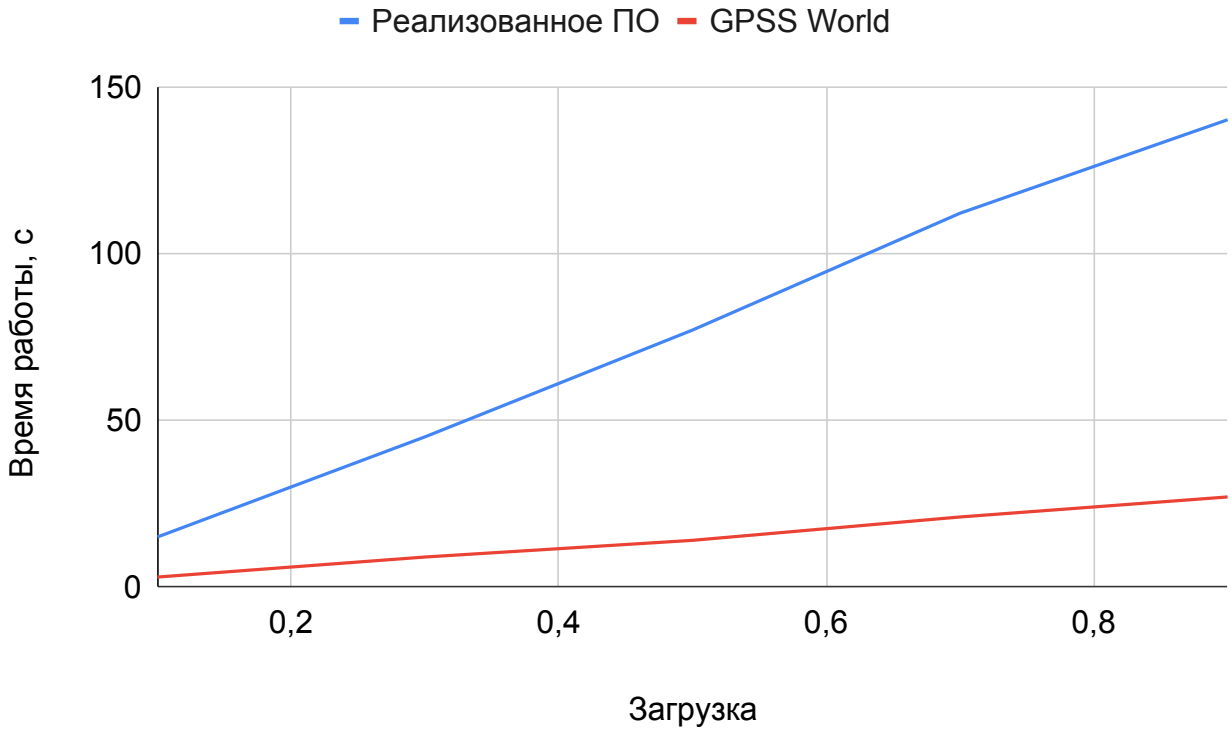

Рисунок 4.5 – Сравнение зависимостей времени работы от загрузки

На рисунке 4.6 изображён график зависимости времени работы от времени моделирования для реализованного ПО и GPSS World.

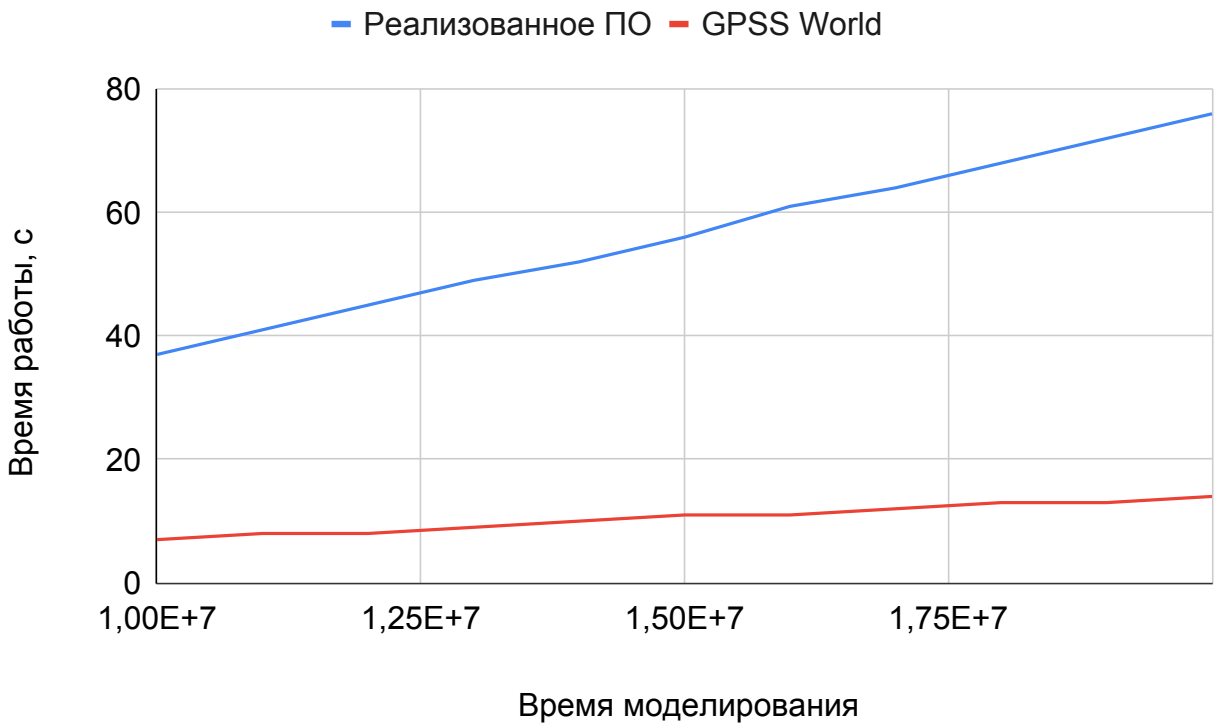

Рисунок 4.6 – Сравнение зависимостей времени работы от времени моделирования

Можно сделать вывод от том, что обе зависимости имею линейный характер, как у реализованного ПО так и GPSS World, но график реализованного ПО растёт быстрее. Также GPSS World в среднем работает в 5 раз быстрее.

4.3 Вывод

Можно сделать вывод, что для рассмотренной модели параметра моделирующего алгоритма почти не оказывают значения на время работы. Также зависимость времени работы реализованного ПО как от загрузки, так и от времени моделирования имеет линейный характер.

По сравнению с GPSS World реализованное ПО работает в среднем в 5 раз медленнее.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Градов В.М. Компьютерное моделирование / В.М. Градов, Г.В. Овечкин, П.В. Овечкин, И.В. Рудаков — М.:КУРС ИНФРА–М, 2019. — 264 С.

2. Посещаемость «Мои документы» в 2022 году [Электронный ресурс]. — URL: `https://www.mos.ru/news/item/117681073/` (дата обращения: 12.11.2023).

3. Расширение концепции ООО–модели для систем массового обслуживания на примере многофункционального центра предоставления государственных и муниципальных услуг / А.В. Чуев, С.А. Юдицкий, В.З. Магергут // Экономика. Информатика. — 2015. — №. 1. – С. 85–93.

4. Пронникова Т.Ю. Применение имитационного моделирования для оптимизации бизнес-процессов обслуживания клиентов в многофункциональном центре / Т.Ю. Пронникова, М.Н. Рассказова // Прикладная математика и фундаментальная информатика. — 2022. — С. 122-123.

5. Сутягина Н. И. Моделирование деятельности многофункционального центра как системы массового обслуживания // Карельский научный журнал. — 2015. — №. 1. — С. 199–203.

6. Моделирование систем / С.П. Бобков, Д.О. Бытев // —Иваново:Изд. ИвГХТУ, 2008. — 156 с.

7. Теория автоматов / Ожиганов А.А. // — СПб.:НИУ ИТМО, 2013. — 84 с.

8. Моделирование систем / Альсова О.К. // — Новосибирск:Изд-во НГТУ, 2007. — 72 с.

9. Блюмин, С.Л. Дискретное моделирование систем автоматизации и управления / С.Л. Блюмин, А.М. Корнеев. — Липецк:ЛЭГИ, 2005. — 124 с.

10. Осипов Г.С. Математическое и имитационное моделирование систем массового обслуживания / Г.С. Осипов — М.: Издательский дом Академии Естествознания, 2017. — 56 с.

11. Григорьева Т. Е., Донецкая А. А., Истигечева Е. В. Моделирование одноканальных и многоканальных систем массового обслуживания на примере билетной кассы автовокзала / Т.Е. Григорьева, А.А. Донецкая, Е.В. Истигечева // Вестник Воронежского института высоких технологий. – 2017. — №. 1. — С. 35–38.

12. Мальков М.В. Сети Петри и моделирование / М.В. Мальков, С.Н. Малыгина // Труды Кольского научного центра РАН. — 2010. — №. 3. — С. 35–40.

13. Мараховский В. Б. Моделирование параллельных процессов. Сети Петри. / В. Б. Мараховский, Л. Я. Розенблюм, А. В. Яковлев — СПб.:Профессиональная литература, 2014. — 400 с.

14. Устимов К. О., Федоров Н. В. Автоматизация построения имитационной модели бизнес-процессов на основе методологии IDEF0 и раскрашенных сетей Петри // Горный информационно–аналитический бюллетень (научно-технический журнал). — 2013. — №. 12. — С. 90–94.

15. Климов А. В. Цветные сети Петри и язык распределенного программирования UPL: их сравнение и перевод // Программные системы: теория и приложения. — 2023. — Т. 14. — №. 4. — С. 91-122.

16. Кельтон В. Имитационное моделирование. Классика CS. 3-е изд. / В. Кельтон, А. Лоу — СПб.:Издательская группа BHV, 2004. — 847 с.

17. Советов Б. Я. Моделирование систем // Б. Я. Советов, С. А. Яковлев — М.: Высш. шк., 2001. — 343 с.

18. Аврамчук Е. Ф. Технология системного моделирования/ Е. Ф. Аврамчук, А. А. Вавилов, С. В. Емельянов и др.; Под общ. ред. С. В. Емельянова и др. — М.: Машиностроение, 1988. — 520 с.