УДК 681.3.016

УЛЬЯНИЦКАЯ К.А.,
КУНЩИКОВ Е.О.,
ОСТРОВСКИЙ С.М.

ПОДХОД К ИСПОЛЬЗОВАНИЮ МАТРИЧНОГО МЕТОДА ДЛЯ АВТОМАТИЗАЦИИ БИЗНЕС ПРОЦЕССОВ

This material is devoted to the use of matrix methods to create and automate business processes. Considered a specific business process "Manufacturing production" and his support through SEDO. Was carried out general analysis of documents prepared by the state diagram of BP formed BZ logical rules movement control documents. To the obtained system of logical rules applied matrix method for the evaluation of the probability of working off for the business processes. Module was designed to optimize the business process. And the comparative analysis was made. A system of logical rules will be used to automate BP "Making products" based on SEDO. The matrix method will be used to determine the probability of working off of documents entering.

Данный материал посвящен использованию матричного метода при создании и автоматизации бизнес процессов. Рассмотрен конкретный бизнес-процесс «Изготовление продукции» и его поддержка на основе Седо. Был проведен общий анализ документооборота организации, составленная диаграмма состояний БП, сформированная БЗ логических правил управления движением документов. К полученной системы логических правил применен матричный метод для оценки достоверности отработки документа для данного БП. Был разработан модуль для оптимизации бизнес процесса. И проведен сравнительный анализ. Полученная система логических правил будет использована для автоматизации БП «Изготовление продукции» на основе Седой. Матричный метод будет использован для определения вероятности отработки документов, поступающих.

Поддержка бизнес-процесса «Изготовление продукции» на основе СЕДО

Необходимо начать с общего анализа документооборота организации, который должен помочь определить набор операций над документами, политику маршрутизации документа на всех этапах его жизненного цикла. В общем случае документооборот организации включает в себя следующие процессы:

- регистрация документа: после получения документа извне или создания внутри, документ регистрируется, т.е. на документ заводится регистрационная карточка, ему присваивается уникальный «идентификатор;
- анализ документа: результатом такого анализа документа будет в простейшем случае ответный документ и переход документа на завершающую стадию, в других же случаях, и их большинство, – результатом анализа будет документ (резолюция), прикрепляющаяся к документу, и являющаяся поручением, которое нужно будет выполнить соответствующим лицам, определенным в резолюции;
- обработка документа: движение документа и его копий; документ/копии документа после анализа чаще всего попадают к исполнителям, которых может быть достаточно большое количество; выполнение документа, естественно, контролируется определенными должностными лицами, что также должно быть учтено при построении системы управления движением документов; при этом схемы движения документа имеют достаточно сложную структуру с циклами, разветвлениями и последовательно выполняемыми операциями;
- завершающая стадия – отправка документа.

Определено три основных состояния: когда документ находиться в канцелярии, у главного технолога и на обработке. Все три состояния являются составными, так как состоят из не скольких подсостояний.

Опишем переходы на диаграмме:

- Зарегистрировать документ;
- Проанализировать документ;
- Если документ исходящий или в результате анализа не требуется его дальнейшая обработка и рассмотрение, то его отправляют по назначению;
- Документ поставлен на контроль;
- На документ наложена резолюция;
- Документ отправлен на обработку;
- Обработать документ;

- Сформировать отчет по отработке документа;
- Согласование с ПЭО;
- Отправка на подпись к главному технологу;
- Снятие с контроля;
- Отправка документа.

Формирование БЗ логических правил управления движением документов

Это наиболее ответственный этап. Неверные правила, отсутствие правил для некоторых ситуаций приводят к неправильным решениям, сбоям в функционировании документооборота. Правила имеют вид клауз Хорна: [1]

$A_1 \leftarrow B_1, B_2, \ldots, B_n$

Подразумевается, что A_1 выполняется, если выполняются условия $B_1, B_2, \ldots, B_n$.

Клаузальная логика – это форма стандартной логики (классической) и отличается от нее системой обозначений. В основе стандартной формы логики лежит логика высказываний (пропозициональная логика) и логика предикатов. При записи предложений в клаузальной форме кванторы в явном виде не указываются, но предполагается, что все переменные связаны квантором всеобщности. В примере выше B1,...,Bm, A1,...,An – это атомарные формулы ($n \geq 0$, $m \geq 0$), A1,...,An – это совместные посылки клаузы, а B1,...,Bm − это альтернативные заключения. В стандартной форме клауза равносильна записи A1 & ...& An → B1 ∨ ... ∨ Bm.

Таким образом, запятые в посылке клаузы означают конъюнктивные связки между предложениями, составляющими посылку, а запятые в заключении клаузы символизируют дизъюнктивные связки между предложениями, составляющими посылку. Множество клауз невыполнимо, если из него можно вывести пустое предложение (обозначаемое). [2]

Наиболее важной характеристикой, используемой в правилах управления движением документов, является состояние документа. Вводим переменную состояния **z,** которая может принимать значения из множества состояний определенных в схеме.

Определим основные состояния документа:

Z1- документ зарегистрирован
Z2- документ проанализирован
Z3- документ поставлен на контроль
Z4- документ рассмотрен, резолюция
Z5.1, Z5.2, Z5.3, Z5.4– документ исполнен соответствующими отделами: отдел снабжения, отдел комплектации, отдел электротехнической подготовки, инструментальное производство.
Z6.1, Z6.2, Z6.3, Z6.4- создан отчет об обработке документа: отдел снабжения, отдел комплектации, отдел электротехнической подготовки, инструментальное производство.
Z7- формирование ответа
Z8- согласование
Z9- документ подписан
Z10- документ снят с контроля
Z11- документ отправлен

Клаузы Хорна нашей системы будут иметь следующий вид, где число рядом с стрелочкой отображает вероятность с какой условие обеспечивает последствие:

1) $|\rightarrow_1 Z1$
2) $|\rightarrow_1 (Z1 \rightarrow Z2)$
3) $|\rightarrow_{0.7} (Z2 \rightarrow Z3)$
4) $|\rightarrow_{0.95} (Z2 \rightarrow Z4)$
5) $|\rightarrow_{0.9} ((Z2 \wedge Z4) \rightarrow (Z5.1 \wedge Z5.2 \wedge Z5.3 \wedge Z5.4))$
6) $|\rightarrow_1 (Z5.1 \rightarrow Z6.1)$
7) $|\rightarrow_1 (Z5.2 \rightarrow Z6.2)$
8) $|\rightarrow_1 (Z5.3 \rightarrow Z6.3)$
9) $|\rightarrow_1 (Z5.4 \rightarrow Z6.4)$
10) $|\rightarrow_1 ((Z6.1 \wedge Z6.2 \wedge Z6.3 \wedge Z6.4) \rightarrow Z7)$
11) $|\rightarrow_{0.9} (Z7 \rightarrow Z8)$
12) $|\rightarrow_{0.95} ((Z7 \wedge Z8) \rightarrow Z9)$
13) $|\rightarrow_1 (Z9 \rightarrow Z10)$
14) $|\rightarrow_1 (Z10 \rightarrow Z11)$

Так как свойства и функции отделов одинаковы, то для упрощения последующих расчетов представляем состояния Z5.1, Z5.2, Z5.3, Z5.4 в общем виде Z5, а Z6.1, Z6.2, Z6.3, Z6.4 в виде Z6.

Для построения атомов логических правил также используются определенные значения атрибутов документа, для учета которых вводим переменные:

- переменная “вид” документа **v,** которая может принимать значения из множества **{V_1(**внутренний документ), **V_2(**входящий документ) и др.**}**;
- переменная “тип” документа **a,** которая может принимать значения из множества **{A_1 (**справка)**,** **A_2 (**поручение)**,** **A_3** (заявка)**,** **A_4 (**приказ) и т.д.**}**;
- переменная “организация” (или человек) откуда документ пришел **o,** которая может принимать значения из множества **{O_1** (НТУ «КПИ»)**,** **O_2** (НАУ)**,** **O_3** (ВАТ «ВаБанк») и т.д.**}**;

- переменная “срочность” документа **c,** которая может принимать значения из множества **{C_1** (срочный)**, C_2** (неотложный)**, C_3**(несрочный)**}**;
- переменная “тематика” документа **t,** которая может принимать значения из множества **{T_1**(финансовый документ)**, T_2**(банковские расчеты), **T_3** (договор) и т.д.**}**.

Рассмотрим примеры правил вывода системы:

Для того, чтобы перевести документ из состояния Z_0 в одно из под-состояний состояния Z_1, необходимо чтобы были выполнены определенные операции и соблюдался ряд условий. Например, для перехода документа в состояние Z_1^1(«Документ зарегистрирован в журнале для входящих общих документов») необходимо, чтобы тип документа был «заявка» ($a=A_3$), организация, из которой он поступил – ВАТ «ВаБанк» ($o=O_3$), документ должен быть срочным ($c=C_1$), а тематика документа – заказ. Для осуществления такого рода действий надо составить соответствующие логические правила:

- Для перехода из состояния Z_0 в состояние «Документ зарегистрирован в журнале для внутренних документов» Z_1^0:

$$(z= Z_1^0) \leftarrow (v=V_1)$$

- Для перехода из состояния Z_0 в состояние «Документ зарегистрирован в журнале для входящих общих документов» Z_1^1:

$$(z= Z_1^1) \leftarrow (a=A_3 , o=O_3 , c=C_1 , t=T_3)$$

- Для перехода из состояния Z_0 в состояние «Документ зарегистрирован в журнале для переходящих документов» Z_1^2:

$$(z= Z_1^2) \leftarrow (a=A_3 , o=O_2 , c=C_3 , t\neq T_2)$$

- Для перехода из состояния Z_0 в состояние «Документ зарегистрирован в журнале для входящих специфических документов» Z_1^3:

$$(z= Z_1^3) \leftarrow (a=A_2 , o=O_1 , c=C_3)$$

- Для перехода из состояния Z_0 в состояние «Документ зарегистрирован в журнале для исходящих документов» Z_1^4:

$$(z= Z_1^4) \leftarrow (a=A_3 , c=C_4)$$

Отобразим схематически один из возможных планов передвижения документа из начального состояния Z_0 в конечное состояние Z_{12} (рис. 1.):

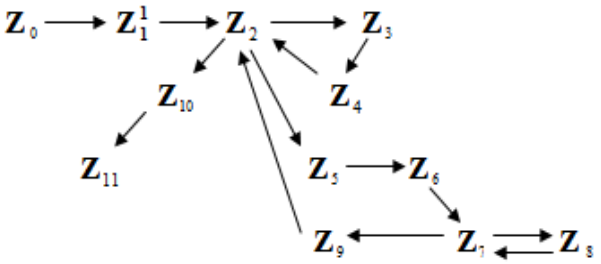

Рис. 1. План (схема) передвижения документа из начального состояния Z_0 в конечное состояние Z_{11}

Переходы между состояниями обозначены направленными стрелками, каждому переходу соответствует определенный набор логических правил.

На рисунке 1 отображен простейший жизненный цикл пришедшего на обработку документа: документ поступил из внешней организации (Z_0), зарегистрирован в соответствующем журнале (Z_1^1), после чего проанализирован (Z_2), после анализа документа он был поставлен на контроль(Z_3) и на него была наложена резолюция (Z_4); после того как документ опять пройдет анализ, в случае необходимости его переслали на исполнение (Z_5), после отработки,в каждом отделе был написан отчет об исполнении документа (Z_6). Затем в отделе кооперации формируется общий отчет(Z_7), который перед тем как пойти на подпись к главному технологу проходит согласование в ПЭО(Z_8). Отчет вместе с документом поступает на подпись вышестоящему начальству (Z_9). Если документ подписан (а в нашем случае это так), то он возвращается на стадию анализа (Z_2), после чего документ снимается с контроля(Z_{10}), и отсылается (Z_{11}).

Реализация метода

Предлагается 2-х этапная схема реализации:

1) Этап вывода. Выводится план (схема) движения документа на основе правил, заложенных в системе. Этот план включает операции, которые предстоит выполнить и условия, которые нужно проверить перед выполнением операции. Поскольку следующее действие зависит от результатов предыдущего, план должен достраиваться в процессе его реализации.

2) Этап осуществления планов. Когда план построен или построена определенная часть плана, необходимо отслеживать условия вы-

полнения операций плана и передавать документ для выполнения соответствующих действий над ним.

Реализацию первого этапа необходимо осуществлять с помощью механизмов вывода. Предлагается рассмотреть матричный метод. [3]

Система должна быть непротиворечивой, то есть у нее должен быть закладений алгоритм анализа правил на противоречивость. Идея заключается в применении матричного представления выходных данных и реализации резолютивного метода в виде операции умножения матриц.

Нечеткий логический вывод

Одной из проблем создания применений есть работа с неопределенностью, неточностью, неполнотой информации. Достаточно часто она бывает нечеткой по своей природе. В классических условиях применяют логико-математические исчисления, причем механизмы выведения лишают алгоритмический подход от некоторых его существенных недостатков. Эффективным и удобным средством для проработки нечетких знаний является нечеткая логика. Предлагая аппарат определения численных значений нечеткости, вычисления выражений с элементами нечеткости, она дает возможность находит решение даже в случае недостаточности данных.

Допустимо, что существует множественное число правил вида:

$$(A1 \wedge A2 \wedge \ldots) \rightarrow (B1 \vee B2 \vee \ldots) \qquad (1)$$

где A1, A2,… – это условия правила, B1, B2,… – возможные последствия.

Для каждого следствия экспертами определена оценка уверенности в том, что он имеет место для выражения, которое удовлетворяет условия правила. Последствия правила могут составлять условия для других правил, и таким образом мы получаем цепочки правил. Каждому следствию определена оценка уверенности в интервале [0,1]. Если условие имеет одно следствие, его оценка уверенности не обязательно равняется 1.

В большинстве случаев существует две и больше возможных последовательностей применения правил выведения. Чтобы определить, которая из них дает конечный результат с большей степенью уверенности, необходим метод вычисления промежуточных результатов на основе степени уверенности условия и степени уверенности того, что определенное условие обеспечит определенное следствие. Другими словами, если известно, что уверенность в A1 равняется к, а уверенность в A1→A2 равняется l, то для последующих вычислений по цепочке или (в случае, если A2 – конечная точка) получения конечной уверенности в целесообразности применения этой цепочки правил необходимо иметь оценку уверенности в A2.

Чтобы получить оценку для A2, обозначим с помощью a′ “не a”, $a \vee b$ – “або”, $a \wedge b$, “і”, $a \rightarrow b$ – импликацию “если a, то b”. В теории нечетких множеств $a' = 1 - a$. Логическому выражению “і” отвечает операция $\otimes$, а выражению “или” – операция $\oplus$. Эти операции определяются через Т-нормы (отражаются знаком *T) и Т-конорми (отражаются знаком +T) соответственно. Самыми распространенными являются такие определения Т-норм и Т-конорм:

$$a *_T b = ab;\ a +_T b = a + b - ab \qquad (2)$$

$$a *_T b = \min\{a, b\};\ a +_T b = \max\{a, b\} \qquad (3)$$

Вычисление выполнять за такой схемой:

Шаг 1. Рассматривать лишь правила вида (1), в каких консеквент содержит не больше одного возможного последствия;

Шаг 2. Если $\vdash_{\rightarrow} к$ A помечает утверждение о том, что A имеет место со степенью уверенности к, то степени уверенности вычислять с помощью формул:

а) если $\vdash_{\rightarrow} k\ P$ і $\vdash_{\rightarrow} l\ Q$, то $\vdash_{\rightarrow} k \otimes l\ P \wedge Q$;

б) если $\vdash_{\rightarrow} k\ P$ і $\vdash_{\rightarrow} l\ Q$, то $\vdash_{\rightarrow} k \oplus l\ P \vee Q$;

в) если $\vdash_{\rightarrow} k\ P$ і $\vdash_{\rightarrow} l\ P \rightarrow Q$, то $\vdash_{\rightarrow} k \otimes l\ Q$;

Шаг 3. Перебор всех возможных цепочек заменить применением матричного метода вывода.

Рассмотрим наш пример:

Сначала применением определения операций (3) и правил выведения шага 2 приведенной выше схемы:

1) вычислим $\vdash_{\rightarrow 1} \otimes_1 Z2$:
   $1 \otimes 1 = \min\{1, 1\} = 1$,
   то есть $\vdash_{\rightarrow 1} Z2$;
2) вычислим $\vdash_{\rightarrow 1} \otimes_{0.7} Z3$:
   $1 \otimes 0.7 = \min\{1,0.7\} = 0.7$,
   то есть $\vdash_{\rightarrow 0.7} Z3$;
3) вычислим $\vdash_{\rightarrow 1} \otimes_{0.95} Z4$:
   $1 \otimes 0.95 = \min\{1,0.95\} = 0.95$,
   то есть $\vdash_{\rightarrow 0.95} Z4$;
4) вычислим $\vdash_{\rightarrow 1} \otimes_{0.95} (Z2 \wedge Z4)$:
   $1 \otimes 0.95 = \min\{1,0.95\} = 0.95$,
   то есть $\vdash_{\rightarrow 0.95} (Z2 \wedge Z4)$;
5) вычислим $\vdash_{\rightarrow 0.95} \otimes_{0.9} (Z5.1 \wedge Z5.2 \wedge Z5.3 \wedge Z5.4)$:

$0.95 \otimes 0.9 = \min\{0.95,0.9\} = 0.9$,
то есть $|\rightarrow_{0.9}$ (Z5.1$\wedge$ Z5.2 $\wedge$ Z5.3$\wedge$ Z5.4);

6) i = 1..4
вычислим $|\rightarrow_{0.9} \otimes_1$ (Z6.i):
$0.9 \otimes 1 = \min\{0.9,1\} = 0.9$,
то есть $|\rightarrow_{0.9}$ (Z6.i);

7) вычислим $|\rightarrow_{0.9} \otimes_{0.9} \otimes_{0.9} \otimes_{0.9}$ (Z6.4$\wedge$ Z6.4$\wedge$ Z6.4$\wedge$ Z6.4):
$0.9 \otimes 0.9 \otimes 0.9 \otimes 0.9 = \min\{0.9,0.9,0.9,0.9\} = 0.9$,
то есть $|\rightarrow_{0.9}$ (Z6.4$\wedge$ Z6.4$\wedge$ Z6.4$\wedge$ Z6.4);

8) вычислим $|\rightarrow_{0.9} \otimes_1$ Z7:
$0.9 \otimes 1 = \min\{0.9,1\} = 0.9$,
то есть $|\rightarrow_{0.9}$ Z7;

9) вычислим $|\rightarrow_{0.9} \otimes_{0.9}$ Z8:
$0.9 \otimes 0.9 = \min\{0.9,0.9\} = 0.9$,
то есть $|\rightarrow_{0.9}$ Z8;

10) вычислим $|\rightarrow_{0.9} \otimes_{0.9}$ (Z7$\wedge$Z8):
$0.9 \otimes 0.9 = \min\{0.9,0.9\} = 0.9$,
то есть $|\rightarrow_{0.9}$ (Z7$\wedge$Z8);

11) вычислим $|\rightarrow_{0.9} \otimes_{0.95}$ Z9:
$0.9 \otimes 0.95 = \min\{0.9,0.95\} = 0.9$,
то есть $|\rightarrow_{0.9}$ Z9;

13) вычислим $|\rightarrow_{0.9} \otimes_1$ Z10:
$0.9 \otimes 1 = \min\{0.9,1\} = 0.9$,
то есть $|\rightarrow_{0.9}$ Z10;

14) вычислим $|\rightarrow_{0.9} \otimes_1$ Z11:
$0.9 \otimes 1 = \min\{0.9,1\} = 0.9$,
то есть $|\rightarrow_{0.9}$ Z11;

Теперь применим матричный метод. Сначала построим матрицу для проверки вывода Z11. Наличие оценок уверенности правил и утверждений требует усовершенствовать процедуру построения матриц. Они, как и для двусмысленной логики, должны иметь столбец и строку для каждого литерала, а элемент на пересечении строки i и столбца j будет иметь не нулевое значение лишь тогда, когда найдется клауза, в безусловной части которой находится литерал строки i, а условная часть содержит литерал столбца j. Но это значение будет нечеткой оценкой. Добавляем клаузу $\leftarrow$ Z11. Учитывая возможность существования нескольких клауз с одинаковой безусловной частью и клауз с условной частью с несколькими литералами (например, сравните $|\rightarrow$ 0.4 A1 $|\rightarrow$ 0.6 (A2 $\rightarrow$ A3) $|\rightarrow$ 0.2 (A1 $\rightarrow$ A3) $|\rightarrow$ 0.7 A2 и $|\rightarrow$ 0.4 A1 $|\rightarrow$ 0.7 A2 $|\rightarrow$ 0.8 (A1,A2 $\rightarrow$ A3)), наличие первого случая будем помечать индексом „$\vee$” возле каждого литерала условной части одной клаузи, а наличие последнего случая - индексом „$\wedge$” возле каждого литерала условной части одной клаузи. Матрица имеет вид:

	◊	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11
◊	0	0	0	0	0	0	0	0	0	0	0	1
Z1	1	0	0	0	0	0	0	0	0	0	0	0
Z2	0	1	0	0	0	0	0	0	0	0	0	0
Z3	0	0	0.7	0	0	0	0	0	0	0	0	0
Z4	0	0	0.95	0	0	0	0	0	0	0	0	0
Z5	0	0	0.9$\wedge$	0	0.9$\wedge$	0	0	0	0	0	0	0
Z6	0	0	0	0	0	1	0	0	0	0	0	0
Z7	0	0	0	0	0	0	1	0	0	0	0	0
Z8	0	0	0	0	0	0	0	0.9	0	0	0	0
Z9	0	0	0	0	0	0	0	0.95$\wedge$	0.95$\wedge$	0	0	0
Z10	0	0	0	0	0	0	0	0	0	1$\wedge$	0	0
Z11	0	0	0	0	0	0	0	0	0	0	1	0

Рис. 2. Матрица M1

Применяя правила A и B приведем матрицу к виду матрицы M1. Модифицируем также алгоритм умножения матриц. В случае наличия правил $|\rightarrow$ к A1 $|\rightarrow$ l (A1 $\rightarrow$ A2), умножение $k \otimes l$ необходимо выполнять за формулой min{к, l}. В случае наличия правил $|\rightarrow$ к A1 $|\rightarrow$ l A2 $|\rightarrow$ q (A1, A2 $\rightarrow$ A3), в матрице в строке появляются индексы „$\wedge$” в столбцах для литералов A1, A2 и умножение $(k \otimes l) \otimes q$ необходимо выполнять за формулой min{min{к, q}, min{l, q}}. В случае наличия правил $|\rightarrow$ к A1 $|\rightarrow$ l A2 $|\rightarrow$ q (A1 $\rightarrow$ A3) $|\rightarrow$ r (A2 $\rightarrow$ A3), в матрице в строке появляются два ненулевих элемента с индексами „$\vee$” в столбцах для литералов A1, A2 и при умножении необходимо выбрать максимальный из двух произведений в строке max {min{к, q}, min{l, r}}.

	◊	Z1	Z2	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11
◊	0	0	0	0	0	0	0	0	0	0	1
Z1	1	0	0	0	0	0	0	0	0	0	0
Z2	0	1	0	0	0	0	0	0	0	0	0
Z4	0	0	0.95	0	0	0	0	0	0	0	0
Z5	0	0	0.9$\wedge$	0.9$\wedge$	0	0	0	0	0	0	0
Z6	0	0	0	0	1	0	0	0	0	0	0
Z7	0	0	0	0	0	1	0	0	0	0	0
Z8	0	0	0	0	0	0	0.9	0	0	0	0
Z9	0	0	0	0	0	0	0.95$\wedge$	0.95$\wedge$	0	0	0
Z10	0	0	0	0	0	0	0	0	1$\wedge$	0	0
Z11	0	0	0	0	0	0	0	0	0	1	0

Рис. 3. Матрица M1

Перемножение матрицы приводит к подтверждению противоречивости исходного множественного числа клауз при p = 10 по признаку противоречивости для традиционной клаузальной логики, которая доводит приведенное выше предположение о логическом следствии Z11.

	◊	Z1	Z2	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11
◊	0.9Λ	0.9Λ	0	0	0	0	0	0	0	0	0.9Λ
Z1	0.9Λ	0.9Λ	0.9Λ	0	0	0	0	0	0	0	0
Z2	0	0.9Λ	0.9Λ	0.9Λ	0	0	0	0	0	0	0
Z4	0	0	0.9Λ	0.9Λ	0.9Λ	0	0	0	0	0	0
Z5	0	0	0.9Λ	0.9Λ	0.9Λ	0.95Λ	0	0	0	0	0
Z6	0	0	0	0	0.9Λ	0.9Λ	0.9Λ	0	0	0	0
Z7	0	0	0	0	0	0.9Λ	0.9Λ	0.9Λ	0	0	0
Z8	0	0	0	0	0	0	0.9Λ	0.9Λ	0.9Λ	0	0
Z9	0	0	0	0	0	0	0.9Λ	0.9Λ	0.9Λ	0.9Λ	0
Z10	0	0	0	0	0	0	0	0	0.9Λ	0.9Λ	0.9Λ
Z11	0.9Λ	0	0	0	0	0	0	0	0	0.9Λ	0.9Λ

Рис. 4. Матрица M10

Теперь остается определиться с нечеткой оценкой результата. Поскольку применением определения операций (3) и правил выведения шага 2 приведенной выше схемы мы получили |→ 0.9 Z11, то нечеткая оценка результата принадлежит диагональному элементу последней матрицы на пересечении строки и столбца Z11.

Проблема использования матричного метода

Существует ряд ограничений и неудобств использования матричного метода. Одну из таких проблем мы рассмотрим. Проблема связана с тем, что при проектировании больших бизнес процессов с использованием матричного метода сама матрица получается очень больших размеров, и пути, получаемые в результате, выходят очень большой длинны. В результате потребуется больше памяти на их хранение, а также, и возможно главное, так как может повлиять на скорость их обработки оператором, это то, что из их внешнего вида очень сложно будет сразу определить сам путь. Представим, что состояния бизнес процесса у нас отмечаются цифрами, а переходы из состояния в состояние стрелкой.

(1 ⟶ 2)(2 ⟶ 3)…(49 ⟶ 64)…(85 ⟶ 73)…

Естественно, имея такой путь, будет очень неудобно его обрабатывать. Поэтому был написан модуль, который позволяет многократно оптимизировать исходный бизнес процесс. Принцип работы модуля состоит в том, чтобы преобразовывать прямые участки в одно целое. То есть если существует участок (1 ⟶ 2)(2 ⟶ 3)(3 ⟶ 4) (4 ⟶ 5) то он заменится на аналогичный только будет иметь вид (1 ⟶ 5) и информация о том из каких частей состоит данный отрезок, будет сохраняться в хранилище данных. Это позволяет сделать схему бизнес процесса более простой и воспринимаемой. Многократная оптимизация означает то, что мы можем применять данный модуль некоторое количество раз на схеме одного бизнес процесса. То есть, оптимизировав исходную схему, опять применим уже к выходной матрице (оптимизированной). Это позволит значительно упростить отображение нашего бизнес процесса. Так же всегда есть возможность получить полный путь по сокращенным. Далее будет рассмотрено применение модуля на нашем бизнес процессе «Изготовление продукции».

Использование модуля оптимизации при расчетах

Принцип работы модуля состоит в том, что прямые участки отработки бизнес процесса представляются как одно целое. То есть если есть участок, который представляет собой определенную последовательность действий, без ветвлений, то можно представить данный участок как начальный и конечный пункт, скрывая промежуточные состояния бизнес процесса. Промежуточные состояния будут сохраняться в хранилище, поэтому всегда будет возможность узнать эти промежуточные состояния и возобновить полную схему бизнес процесса. Выше был произведен расчет бизнес процесса «Изготовление продукции» без применения данного модуля. Рассмотрим его применение на данном бизнес процессе и сравним полученные результаты.

Исходная матрица, отображающая наш бизнес процесс, имеет следующий вид (рисунок 5).

	◊	Z1	Z2	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11
◊	0	0	0	0	0	0	0	0	0	0	1
Z1	1	0	0	0	0	0	0	0	0	0	0
Z2	0	1	0	0	0	0	0	0	0	0	0
Z4	0	0	0.95	0	0	0	0	0	0	0	0
Z5	0	0	0.9Λ	0.9Λ	0	0	0	0	0	0	0
Z6	0	0	0	0	1	0	0	0	0	0	0
Z7	0	0	0	0	0	1	0	0	0	0	0
Z8	0	0	0	0	0	0	0.9	0	0	0	0
Z9	0	0	0	0	0	0	0.95Λ	0.95Λ	0	0	0
Z10	0	0	0	0	0	0	0	0	1Λ	0	0
Z11	0	0	0	0	0	0	0	0	0	1	0

Рис. 5. Исходная матрица, отображающая бизнес процесс

Применив модуль оптимизации один раз на исходной схеме, мы получим следующий вид матрицы (рисунок 6).

	◊	Z2	Z7	Z11
◊	0	0	0	1
Z2	1	0	0	0
Z7	0	0.9Λ	0	0
Z11	0	0	0.9Λ	0

Рис. 6. Оптимизированная матрица

После преобразования видно, что остались те узлы, в которых происходит ветвление, а прямые участки представляются единым целым. Ниже показано, каким образом произошло преобразование.

(◊ - Z2) : Path (◊ - Z1)(Z1 – Z2)

(Z2 – Z7) : Path (Z2 – Z4)(Z4 – Z5)(Z5 – Z6)(Z6 – Z7)

(Z2 – Z7) : Path (Z2 – Z5)(Z5 – Z6)(Z6 – Z7)

(Z7 – Z11) : Path (Z7 – Z8)(Z8 – Z9)(Z9 – Z11)(Z10 – Z11)

(Z7 – Z11) : Path (Z7– Z9)(Z9 – Z10)(Z10 – Z11)

(Z11 - ◊) : Path (Z11 - ◊)

Располагая такими данными, в любой момент, возможно, будет отобразить исходный вид матрицы. Как видно, результат оптимизации значительно упростил существующую матрицу. Сохраняя все исходные данные мы получаем матрицу, которая значительно отличается по размерам от начальной. Во – первых, это позволит сэкономить память, необходимая для хранения данных, так как видно, что размеры матриц отличаются практически в три раза. Во – вторых, при отображении пути движения документа, схема становится более читаемой, понятной и прозрачной. И трете, самое главное, полученные данные позволят значительно ускорить дальнейший расчет для анализа бизнес процесса. Так как расчет связан с перемножением матриц, а при их больших размерах, данная задача становиться ресурсоемкой, требуется большее количество памяти на расчеты, и значительно большее количество времени на обработку информации.

Теперь применим наши полученные результаты после оптимизации к нашим расчетам. Собственно проделаем ту же процедуру что и в основном примере только с оптимизированной матрицей. Матрица для проверки вывода Z11 это и будет наша матрица на рисунке 5.

	◊	Z2	Z7	Z11
◊	0	0	0	1
Z2	1	0	0	0
Z7	0	0.9Λ	0	0
Z11	0	0	0.9Λ	0

Рис. 7. Матрица 1

	◊	Z2	Z7	Z11
◊	0	0	0.9Λ	0
Z2	0	0	0	1
Z7	0.9Λ	0	0	0
Z11	0	0.9Λ	0	0

Рис. 8. Матрица 2

	◊	Z2	Z7	Z11
◊	0	0.9Λ	0	0
Z2	0	0	0.9Λ	0
Z7	0	0	0	0.9Λ
Z11	0.9Λ	0	0	0

Рис. 9. Матрица 3

	◊	Z2	Z7	Z11
◊	0.9Λ	0	0	0
Z2	0	0.9Λ	0	0
Z7	0	0	0.9Λ	0
Z11	0	0	0	0.9Λ

Рис. 10. Матрица 4

Перемножение матрицы приводит к подтверждению противоречивости исходного множественного числа клауз при p = 4 по признаку противоречивости для традиционной клаузальной логики, которая доводит приведенное выше предположение о логическом следствии Z11.

Теперь остается определиться с нечеткой оценкой результата. Поскольку применением определения операций (3) и правил выведения шага 2 приведенной выше схемы мы получили |→ 0.9 Z11, то нечеткая оценка результата принадлежит диагональному элементу последней матрицы на пересечении строки и столбца Z11.

Как видно, по сравнению с результатами, полученными при обычных расчетах, и количество информации и количество шагов расчетов значительно отличаются. При обычных расчетах потребовалось провести 10 шагов с матрицей размером 11 на 11, а во втором случае 4 шага с матрицей размером 4 на 4. Так как наш пример не является полным отображением ре-

ального бизнес процесса и полученные результаты являются довольно удовлетворительными, то можно представить какую пользу может принести данный модуль при расчетах реальных бизнес процессов, где размеры матриц смогут составлять 50 на 50, а то и гораздо больших размеров.

Реализация подхода

Программная реализация механизмов оптимизации и расчетов разработана с помощью среды разработки Microsoft Visual Studio 2010 + .Net Framework 4.0 и языка программирования C#. Отображение было разработано с помощью технологии WPF(Windows Presentation Foundation) и языка разметки xaml. При написании данного модуля использовался шаблон проектирования MVP (MODEL – VIEW - PRESENTER).[4][5] Суть паттерна в том, чтобы форма (View) реализовывала интерфейс, с которым сможет работать представление данных (Presenter), которое в свою очередь оперирует объектами предметной области (Model). Таким образом, логика приложения отделяется от представления данных. Также, появляется возможность протестировать эту логику.

Presenter выступает в роли связующего модели и представления. По сути это самое сложное место так как View и Model это две отдельные сущности а цель Presenter связать их. Данный подход позволяет создавать абстракцию представления. Реализовать данный паттерн можно при помощи вынесения интерфейсов представления. У каждого представления будут интерфейсы с определенными наборами методов и свойств, необходимых презентеру, презентер в свою очередь инициализируется с данным интерфейсом, подписывается на события представления и по необходимости подсовывает данные.

В нашем модуле вся логика для оптимизации и расчетов была вынесена в отдельный компоновочный блок (библиотеку), который и есть Model в данном случае. View был создан ради демонстрации данного модуля. В этом и есть основное преимущество применения паттерна MVP, так как можно легко и четко разграничить логику и отображение и в случае необходимости подменять эти блоки. То есть для интегрирования данного модуля достаточно подменить или отключить View, если модуль необходимо использовать для промежуточных расчетов, не требующих отображения.

Библиотека называется CalculationLibrary.dll. Данная библиотека предоставляет два интерфейса для работы с ней:

- IMultiplyMatr
- IOptimize

По названию данных интерфейсов легко можно понять их предназначение, интерфейс IMultiplyMatr предназначен для произведения расчетов над матрицами, а интерфейс IOptimize предназначен оптимизаций матриц. Классы, реализующие данные интерфейсы соответственно:

- MultiplyMatr
- Optimize

Данные классы предоставляют все необходимые методы и свойства для реализации данного механизма.

Интерфейс IOptimize предоставляет 4 основные методы и одно свойство для работы:

- FillDefaulutGrid - метод
- OptimizeTable - метод
- GenerateFile - метод
- GetFileData – метод
- Iteration - свойство

Основным является, конечно, метод OptimizeTable. Метод предназначен для оптимизации матрицы. В качестве аргумента принимает объект типа DataTable, отображающий исходную матрицу, которую необходимо оптимизировать, и в качестве возвращаемого значения выступает тоже объект DataTable, содержащий уже новую оптимизированную матрицу.

Остальные методы написаны специально для демонстрации модуля и вряд ли будет необходимость их использования при внедрении, в какие либо системы.

Метод FillDefaulutGrid предназначен для заполнения матрицы значениями по умолчанию, то есть нулями. В качестве аргумента принимает целое число и в результате выполнения метода будет возвращен объект типа DataTable, содержащий матрицу заполненную нулями и размером, который был указан в качестве входного аргумента в метод.

Метод GenerateFile предназначен для генерации файла с данными. Для автоматизации ввода данных в матрицу. При выполнении данного метода на жестком диске в каталоге с исполняемым файлом программы создается файл

Settings.xml, содержащий множество объектов Point, каждый из которых содержит три свойства:

- Col – колонка;
- Row – рядок;
- Probability – вероятность.

При последующей загрузки файла xml каждый объект Point займет свое место в матрице по данным свойствам колонки и рядка, и иметь значение вероятности, которое отобразиться пользователю.

Метод имеет один входной целочисленный аргумент, который означает, сколько объектов Point будет содержать данный файл. В качестве возвращаемого значения методом выступает переменная типа bool (булевый тип), принимающая значение true если удалось создать файл и false если нет.

Метод GetFileData предназначен для считывания матрицы из файла. Не содержит входных аргументов. В качестве возвращаемого значения выступает объект типа DataTable, содержащий матрицу заполненную значениями из файла.

Последним членом данного интерфейса является свойство Iteration. Данное свойство целочисленного типа. На каждой итерации происходят записи в базу данных и при этом в каждой записи есть поле итерация, что бы впоследствии можно было возобновить данные на каждой итерации. Это и есть основное предназначение данного свойства, инкрементировать его на каждой операции оптимизации и записывать в базу с данными.

Интерфейс IMultiplyMatr предоставляет всего лишь два метода:

- Multiply
- ShowExcel

Первый метод Multiply является основным методом данного класса, и предназначен для перемножения двух матриц. Метод имеет два входных аргумента, в качестве которых выступают два экземпляра класса DataTable, содержащие исходные матрицы, необходимые для перемножения. Возвращаемым значением метода является также экземпляр класса DataTable, содержащий уже готовую перемноженную матрицу.

Второй метод ShowExcel предназначен для отображения данных в Excel. Для этого используется COM библиотека Microsoft.Office.Interop.Excel, которая предоставляет необходимый набор методов и свойств для работы с пакетом Microsoft.Excel. Метод предполагается для отображения матрицы полученной в результате перемножения двух других. В качестве входного аргумента является экземпляра класса DataTable, который и содержит необходимую для отображения матрицу. Отображаемый пользователю файл Excel, сгенерированный в результате выполнения метода, содержит не только матрицу, полученную в качестве аргумента в метод и отображающую вероятности перехода из одного состояния бизнес процесса в другой, но и матрицу отображающую пути перехода. Пример представления данных на рисунке 11.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	0	1	1	0	1	1	0		0	(1->6)(6->0)	(2->6)(6->0)	0	(4->6)(6->0)	(5->6)(6->0)	0
2	0	0	0	0	0	0	1		0	0	0	0	0	0	(6->0)(0->1)
3	1	0	0	0	0	0	0		(0->1)(1->2)	0	0	0	0	0	0
4	1	1	0	0	0	0	0		(0->1)(1->3)	(1->2)(2->3)	0	0	0	0	0
5	1	1	1	0	0	0	0		(0->1)(1->4)	(1->3)(3->4)	(2->3)(3->4)	0	0	0	0
6	1	1	1	1	0	0	0		(0->1)(1->5)	(1->4)(4->5)	(2->3)(3->5)	(3->4)(4->5)	0	0	0
7	1	1	0	1	1	0	0		(0->1)(1->6)	(1->5)(5->6)	0	(3->5)(5->6)	(4->5)(5->6)	0	0

Рис. 11. Пример отображения данных в Excel

Существует еще ряд классов, как видно из диаграммы, необходимых для работы модуля. Класс OptimNode является основным классом, над объектами которого происходят различные вычисления, и является отображением ячейки таблицы. Состоит из четырех свойств и одного метода. Свойства:

- Num – свойство предназначено для отображает текущего состояния бизнес процесса.
- NodeToGo – предназначено для хранения значения связанного, дочернего состояния с текущим.
- Path – так как при оптимизации прямые участки графа преображаются в одно целое, а промежуточные состояния скрываются, то весь путь, включая скрытые состояния, между двумя точками хранится в данном свойстве.
- Probability – в данном свойстве хранится вероятность перехода между состояниями. Поле типа double. Может принимать значение от 0 до 1.

Классы Settings, SettingsBuilder, Coordinates – принимают участие при формировании файлов со сгенерированными значениями, отображающие бизнес процесс. При создании файла и при считывании данных из него используется технология сериализации. [6]

Сериализация (в программировании) – процесс перевода какой-либо структуры данных в последовательность битов. Обратной к опера-

ции сериализации является операция десериализации – восстановление начального состояния структуры данных из битовой последовательности.

Сериализация используется для передачи объектов по сети и для сохранения их в файлы. Например, нужно создать распределённое приложение, разные части которого должны обмениваться данными со сложной структурой. В таком случае для типов данных, которые предполагается передавать, пишется код, который осуществляет сериализацию и десериализацию. Объект заполняется нужными данными, затем вызывается код сериализации, в результате получается, например, XML-документ. Результат сериализации передаётся принимающей стороне, например, по электронной почте или HTTP. Приложение-получатель создаёт объект того же типа и вызывает код десериализации, в результате получая объект с теми же данными, что были в объекте приложения-отправителя. По такой схеме работает, например, сериализация объектов через SOAP в Microsoft .NET.

Сериализация предоставляет несколько полезных возможностей:

- Метод реализации сохраняемости объектов, который более удобен, чем запись их свойств в текстовый файл на диск и повторная сборка объектов чтением файлов;
- метод осуществления удалённых вызовов процедур, как, например, в SOAP;
- метод распространения объектов, особенно в технологиях компонентно-ориентированного программирования, таких как COM и CORBA;
- метод обнаружения изменений в данных, изменяющихся со временем.

В нашем случае используется xml сериализация. Класс Settings помечается атрибутом XmlRoot – получает или задает объект, задающий сериализацию с помощью XmlSerializer для класса как корневого элемента XML. Единственное свойство класса Settings – ColumnList помечается атрибутом XmlElement, это означает, что значения данного свойства будут формировать записи в файле XML. ColumnList является коллекцией экземпляров класса Coordinates. В сою очередь свойства класса Coordinates: Col, Row, Probility – помечаются атрибутами XmlAttribute. Это означает, что именно эти три свойства будут отображаться в каждой записи XML файла. Формат записи на рисунке 12.

```
<Point Col="0" Row="0" Probability="0" />
<Point Col="0" Row="1" Probability="1" />
```

Рис. 12. Формат хранения данных в xml

Оставшийся класс SettingsBuilder и занимается сериализацией, то есть формированием и занесением данных в xml файл, десериализацией, то есть вычиткой данных их файла. Для этого в классе предоставляется два метода:

- Serialize – данный метод имеет один входной аргумент, в качестве которого выступает экземпляр класса Settings, который и будет сериализован в файл. Данный метод не имеет возвращаемого значения.
- Deserialize – метод предназначен для считывания данных из XML файла. Не имеет входных аргументов. В качестве возвращаемого значения метода выступает экземпляр класса Settings.

На диаграмме классов остался еще один класс о котом следует рассказать и им является NodeCache. Данный класс является кэшем для хранения состояния бизнес процесса. Для того чтобы не обращаться каждый раз к базе данных при расчетах и создан такой класс буфер. Данные запишутся в кэш при первом обращении, и при последующей необходимости будут использовать данный класс. При проектировании данного класса – кэша использовался шаблон проектирования SingleTon.[7]

Основная его задача – гарантировать, что класс будет иметь лишь единственный экземпляр и глобальную точку доступа к нему. В программировании часто встречаются ситуации, когда необходимо создать класс, который будет существовать только таким образом. Глобальные переменные здесь не годятся, поскольку дают доступ к классу, но не могут запретить его инстанцирование в нескольких экземплярах. В случае работы с шаблоном Singleton сам класс контролирует наличие единственного своего экземпляра и предоставляет доступ к нему.

Итак, Singleton используют в следующих случаях:

- когда необходимо иметь лишь один экземпляр какого-либо конкретного класса, доступного любым клиентам;
- когда единственный экземпляр данного класса способен расширяться через порождения подклассов, а клиенты имеют возможность работать с уже расширенным экземпляром, не внося никаких изменений в свой код.

У класса, реализующего шаблон Singleton, присутствует операция Instance, требующаяся

для того, чтобы предоставить доступ к его единственному экземпляру. Надо понимать, что клиенты получают доступ к экземпляру данного класса только через операцию Instance и никак иначе.

Рассматриваемый паттерн обладает несомненными достоинствами. Выше уже упоминалось, что класс, реализующий шаблон Singleton, инкапсулирует свой единственный экземпляр. Именно благодаря этому он полностью контролирует то, каким образом и когда клиенты будут получать доступ к нему. Кроме того, Singleton позволяет избежать использования глобальных переменных, а значит, не придется засорять ими пространство имен для хранения уникальных экземпляров классов.

Довольно легко создавать и подклассы от класса, реализующего Singleton. Само приложение допустимо сконфигурировать уже расширенным классом. Естественно, можно будет конкретизировать приложение определенным экземпляром класса, нужным в данный момент для выполнения.

Также несложно изменить класс, реализующий Singleton, и предоставить ему возможность создавать более одного экземпляра класса. Это порождает некоторую гибкость, которая всегда может пригодиться в будущем.

К минусам данного шаблона проектирования стоит отнести лишь то, что глобальные объекты, как таковые, могут быть вредны для программы, поскольку препятствуют ее масштабируемости.

Анализ производительности

Производительность данного модуля так же является немало важной частью роботы, так как в случае длительного времени потраченного на расчеты, использование системы становится просто не актуальным. Ни один оператор, ни одна система не захочет использовать данный модуль в случае, если на оптимизацию и анализ требуются 10 минут, а то и десятки.

Проанализируем производительность модуля в зависимости от объема данных, над которыми производятся операции.

1) Анализ оптимизации.

Табл. 1. Данные производительности при оптимизации

Кол-во состояний бизнес процесса.	10	50	100	150
Время потраченное на оптимизацию	36 Млсек.	661 Млсек.	20 Сек	2 мин 22сек

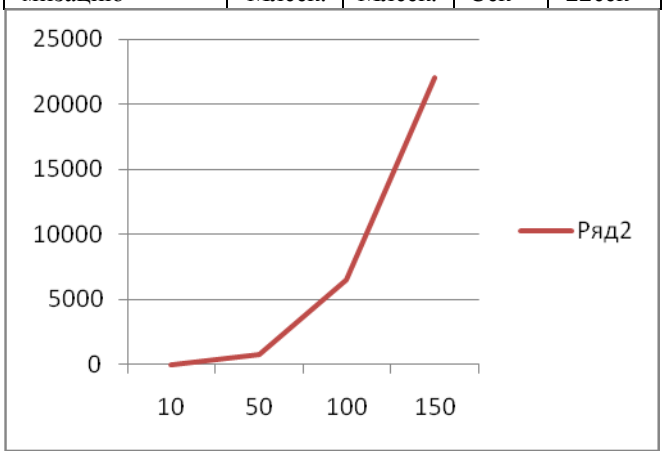

Рис. 13. График производительности по оптимизации

На рисунке 13 отображено график производительности системы при оптимизации бизнес процесса. По горизонтальной оси показано количество состояний, по вертикальной оси показано количество миллисекунд необходимые для оптимизации матрицы. Как видно зависимость имеет экспоненциальный вид. И говорит о том, что при изменении количества состояний бизнес процесса – количество времени необходимое для оптимизации увеличивается в разы.

2) Анализ расчетов.

Табл.1. Данные производительности при расчетах

Кол-во состояний бизнес процесса.	10	50	1	150
Время потраченное на расчеты	24 Млсек.	817 Млсек.	6.5 Сек	22 Сек

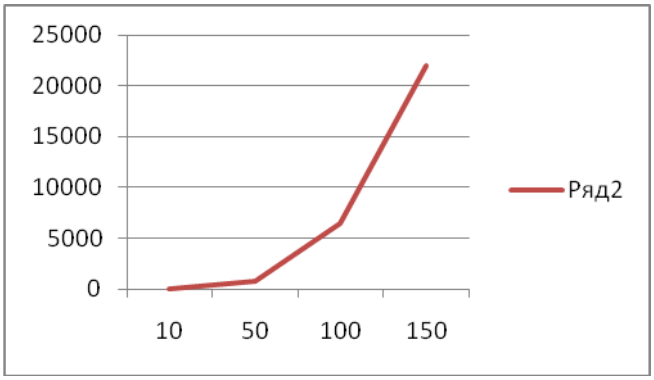

Рис. 14. График производительности по расчетам

На рисунке 14 отображено график производительности системы при расчетах, анализе бизнес процесса. По горизонтальной оси показано количество состояний, по вертикальной оси показано количество миллисекунд необходимые для расчетов матрицы. Как видно с те-

кущего графика зависимость так же имеет экспоненциальный вид, но рост графика не такой значительный как при оптимизации.

Данные исследования полностью не отображают реальные данные по производительности, так как для расчетов и оптимизации использовались бизнес процессы сгенерированные приложением, а не настоящие, имеющие практическое применение. Бизнес процесс, полученный с помощью приложения, представляет собой схему, в которой практически каждое состояние имеет связь с остальными. Из-за этого прямых участков в схеме остается малое количество, и оптимизация не приносит большой выгоды. В такой ситуации, когда у графа почти все состояния связаны друг с другом, сама суть использования бизнес процессов теряется. В реальной жизни такие процессы имеют более простой вид, более прямолинейные зависимости, что делает применение модуля очень полезным, и это было продемонстрировано на примере бизнес процесса «Изготовление продукции».

При использовании данного модуля на бизнес процессе «Изготовление продукции» во время оптимизации было достигнуто значение практически в 65%, так как исходная схема содержала 11 состояний, а оптимизированная – 4. Вследствие чего для расчетов матрицы соответствующего размера необходимо было перемножить 10 раз для исходной матрицы, и 3 раза для оптимизированной.

Время потраченное на оптимизацию составляет 45 миллисекунды. На расчет одной итерации 17 миллисекунды. Учитывая то, что необходимо произвести 3 итерации общее время составляет:

T = 3* 17 + 45 = 96 миллисекунды

Время потраченное на одну итерацию расчетов исходной матрицы составляет 27 миллисекунды. Общее время на расчеты составляет:

T = 27 * 10 = 270 миллисекунд

Учитывая полученные данные, можно получить график производительности для данного реального бизнес процесса (рисунок 15).

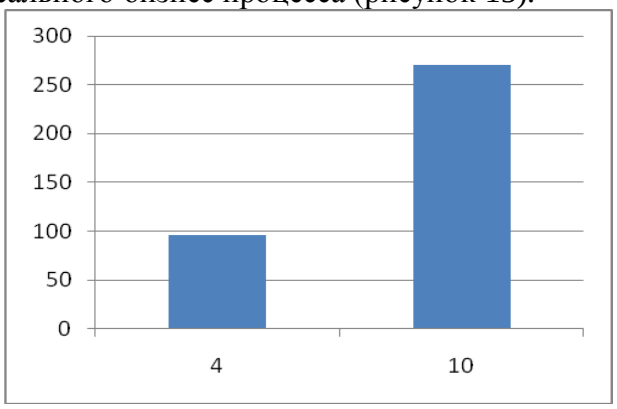

Рис. 15. График производительности бизнес процесса «Изготовление продукции»

Из графика видно разницу в производительности при использовании модуля оптимизации и без него. Во-первых, выигрыш во времени очевиден. Во – вторых, данные полученные в результате расчетов представляются в удобном виде для пользователя.

Выводы

Настоящий материал посвящен проблеме создания системы построения бизнес-процессов посредством реализации схем (планов) действий на основании системы бизнес-правил для автоматизированных систем управления любого уровня технической и программной насыщенности с использованием матричного метода.

Главными преимуществами внедрения модуля является возможность анализа составленных бизнес процессов, получения вероятности отработки конкретных БП конкретными путями, а также решена проблема отображения путей бизнес процессов.

Список литературы

1 Ершов Ю.Л., Палютин Е.А. Математическая логика: Учебное пособие для вузов. Изд.2. 1987. – 336 с.
2 Математическая логика и теория алгоритмов: Учебное пособие. - 2-е изд.120-125стр.
3 Теленик С.Ф., Амонс А.А, Смічик Р.В., Хмелюк В.С. Матрична резолюція для клаузальних логік // Вестник ХНАДУ. Сб. науч. тр. Вып.28, 2003. – С.243-246.
4 http://www.martinfowler.com/eaaDev/PresentationModel.html by Martin Fowler
5 http://msdn.microsoft.com/msdnmag/issues/06/08/DesignPatterns/default.aspx by Jean-Paul Boodhoo
6 Эндрю Троелсен «Язык программирования C# 2005 и платформа .Net 2.0». Глава 17.
7 Дмитрий Федоров “Примеры использования Паттерн Singleton (Одиночка)”