КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2013 Т. 5 № 1 С. 25−36

Ки&М

МОДЕЛИ В ФИЗИКЕ И ТЕХНОЛОГИИ

УДК: 517.977.5

Подсистема «Разработчик» системы приема коммунальных платежей

А. А. Клименко[a], Г. А. Угольницкий

Южный федеральный университет,
факультет механики, математики и компьютерных наук,
Россия, 344090, г. Ростов-на-Дону, ул. Мильчакова, 8а

E-mail: [a] antklim@gmail.com

Получено 20 июля 2012 г.

В работе рассматривается одна из ключевых подсистем приема коммунальных платежей «Разработчик». Описана разработанная система массового обслуживания, которая моделирует данную подсистему. Поставлена и решена задача о распределении ресурсов (в решении использовался модифицированный «венгерский» алгоритм). Приведено описание имитационной (агентной) модели данной подсистемы и результаты имитационных экспериментов.

Ключевые слова: система массового обслуживания, задача о распределении, модификация «венгерского» алгоритма, имитационное моделирование, агентная модель, имитационный эксперимент

Subsystem “Developer” as a part of the Retail Payment System

A. A. Klimenko, G. A. Ougolnitsky

Southern Federal University, Department of Applied Mathematics and Computer Science, 8a Milchakova str., Rostov on Don, 344090, Russia

Abstract. — In this paper we consider one of the core subsystems of the retail payment system named “Developer”. The Queuing System for modeling this subsystem was developed and information about it is provided. The task for the assignment problem was set up and solved (the modification of the Hungarian algorithm was used). Information about Agent Based Model for subsystem “Developer” and the results of the simulation experiments are given.

Keywords: queuing system, assignment problem, Hungarian algorithm, simulation modeling, agent based model

Citation: *Computer Research and Modeling*, 2013, vol. 5, no. 1, pp. 25–36 (Russian).

Введение

В последнее время бизнес теснее стал взаимодействовать с такой областью науки, как информационные технологии. Параллельно с внедрением информационных технологий в бизнес стала бурно развиваться область моделирования и прогнозирования бизнес-процессов, что определяет актуальность темы статьи.

Существует ряд работ, посвященных моделированию бизнес-процессов. Согласно [Vergidis et al., 2008], методы моделирования бизнес-процессов можно разделить на три типа: диаграммные модели, математические модели и языки моделирования бизнес-процессов. Первый тип – это простые диаграммы, изображающие бизнес-процесс, в большинстве случаев без использования стандартных обозначений [Havey, 2005]. Фалп и Шепперд [Phalp, Shepperd, 2000] отметили недостаток данного типа моделирования – анализ на основе использования этих моделей во многом зависит от квалификации аналитика. Второй тип соответствует моделям, имеющим математическое или иное формальное основание. Преимуществом таких моделей является то, что они могут быть проверены математически на предмет логичности и других свойств [Koubarakis, Plexousakis, 2002]. Недостатки формальных методов рассмотрены в работе [Hofacker, Vetschera, 2001]. Третий тип содержит искусственные языки, поддерживающие моделирование бизнес-процессов. К ним относятся BPMN [White, 2004], UML и UML2 [Джейсоул, 1973], XPDL [Елиферов, 2006], YAML [Aalst, Hofstede, 2003, Havey, 2005].

По теме имитационного моделирования известны работы Шрайбера Т. Дж. [Шрайбер, 1980], Борщева А. В. [Борщев,2004], Карпова Ю. Г. [Карпов, 2005] и др.

В настоящей статье рассмотрена подсистема «Разработчик», которая входит в состав системы приема коммунальных платежей. Подсистема «Разработчик» является незаменимой, так как от нее зависит качество работы системы в целом. Для ее поддержания выделена отдельная группа программистов, численность которой может быть произвольной.

Система приема коммунальных платежей функционирует в ОАО КБ «Центр-Инвест» с 2008 года и представляет собой приложение с клиент-серверной архитектурой. В качестве СУБД используется IBM Informix, клиентская часть реализована в виде WEB-приложения на WEB-сервере Apache.

Целью работы является построение и исследование математических моделей подсистемы «Разработчик». Для достижения цели решены следующие задачи: формализация подсистемы «Разработчик», сбор и анализ данных о работе подсистемы, постановка задачи оптимизации, построение имитационной модели, выработка практических рекомендаций, направленных на улучшение производительности исследуемой подсистемы.

Анализ системы приема коммунальных платежей

Система приема коммунальных платежей работает в ОАО КБ «Центр-Инвест» на протяжении трех лет. Платежи принимаются в коммерческом банке. Пункты приема платежей рассредоточены по всему городу. Кассиры осуществляют прием наличности (оплата услуг связи, коммунальных услуг и т. п.) от клиентов круглосуточно. Согласно закону РФ о проведении банковских операций, деньги из кассы не могут быть напрямую перечислены поставщику услуг, требуется постобработка платежей, принятых кассиром. Поэтому система приема коммунальных платежей включает в себя модуль работников бэк-офиса, которые осуществляют постобработку платежей.

Ввиду сложности формализации процесса приема платежей и постобработки (связано с ограничениями, накладываемыми налоговым законодательством и нормативами ЦБ РФ) некоторые операции либо разрешение проблемных ситуаций проводятся вручную или с использованием специальных режимов с максимальным уровнем доступа. Все специальные режимы объединены в отдельный модуль администрирования.

При детальном рассмотрении общей системы приема и обработки коммунальных платежей в ее составе можно выделить подсистемы массового обслуживания, описывающие работу следующих процессов:

1. процесс приема платежа;
2. процесс обработки информации о платежах (например, процессы порождения платежных документов в централизованной базе данных платежей);
3. процесс поддержки пользователей при работе с прикладным программным обеспечением;
4. процесс разработки дополнительных модулей прикладной программы и оптимизации существующих.

Указанные подсистемы соответственно можно назвать «Кассир», «Центральная база данных», «Администратор» и «Разработчик». В данной статье подробно рассмотрена подсистема «Разработчик».

Система массового обслуживания «Разработчик»

Над системой работают от одного до *N* разработчиков. Задания, поступающие к разработчикам, характеризуются по нескольким критериям. Первый из них – тип задачи. Можно выделить три типа задач: исправление ошибок в существующем ПО, разработка новых функций, изменение существующих функций. В зависимости от типа задачи определяется приоритет ее выполнения.

С другой стороны, поступающие задачи различаются по степени их сложности либо объему работ. Объединение этих двух факторов обосновано тем, что единицей измерения работ являются человеко-часы, потраченные на задачу.

На основе вышеизложенного определим основные характеристики системы массового обслуживания, а затем определим, к какому типу она относится. Очевидно, что входным потоком здесь является поток заявок на разработку/доработку ПО. Обслуживающим прибором здесь выступает разработчик. Выходным потоком являются функционирующие части ПО, которые подверглись изменениям. Пусть D – непустое множество приборов, $D = \{d_i, i = 1, ..., n\}$. Каждый разработчик имеет накопитель заявок неограниченной емкости. Таким образом, возникает сеть массового обслуживания с N обслуживающими приборами.

Рассмотрим природу входного потока системы. Так как существует несколько типов задач, то имеем дело с неординарным входным потоком. Пусть в системе имеется $j = 1, ..., H$ типов заявок. Так как интервалы времени между соседними заявками любого *j*-го класса являются случайными величинами, то получаем H случайных потоков. Пусть λ_{ij} — интенсивность потока заявок *j*-го типа в *i*-м обслуживающем приборе системы. Так как λ_{ij} не меняется во времени, то имеем дело со стационарным потоком. Для доказательства неизменности λ_{ij} были рассмотрены статистические данные, собранные за два года. Каждый год разобьем на 12 интервалов. В таблице 1 представлена интенсивность потока заявок в разные годы.

Таблица 1. Интенсивность потока заявок в разные годы

Мес.	1-й год	2-й год
1	10	12
2	7	6
3	6	7
4	6	5
5	4	4
6	3	3
7	3	2
8	3	2
9	4	5
10	5	5
11	9	10
12	12	10

Вычислим среднюю интенсивность по каждому году. Получим значения 6 и 5,92.

Предположим, что интенсивность заявок неизменна, т. е. имеются незначительные отклонения от среднего значения при уровне значимости $\alpha = 0,05$. Для

этого вычислим

$$\chi^2_{эмп1} = \sum_{i=1}^{12} \frac{(n_{i1} - s_1)^2}{s_1} \approx 16{,}3\,, \quad \chi^2_{эмп2} = \sum_{i=1}^{12} \frac{(n_{i2} - s_2)^2}{s_2} \approx 19{,}77\,,$$

где n_{i1} — интенсивность в i-м месяце первого года, n_{i2} — интенсивность в i-м месяце второго года, s_1 — среднее значение интенсивности первого года, s_2 — среднее значение интенсивности второго года.

По таблице критических распределений χ^2 по заданному уровню значимости $\alpha = 0{,}05$ и числу степеней свободы k = 12 — 1 находим критическую точку $\chi^2_{кр} = (0{,}05; 12) = 21{,}0$. Поскольку $\chi^2_{эмп1} = 16{,}3 < 21{,}0 = \chi^2_{кр}$, $\chi^2_{эмп2} = 19{,}77 < 21{,}0 = \chi^2_{кр}$, то нет оснований отвергать гипотезу об одинаковой интенсивности заявок.

Рассмотрим подробнее значения λ_{ij} для каждого месяца первого года и покажем, что они неизменны. Пусть $i, j = 1 \ldots 3$. Рассмотрим 12 матриц размерности 3*3:

$$\begin{bmatrix} 2&0&1 \\ 0&3&1 \\ 1&0&2 \end{bmatrix}, \begin{bmatrix} 2&0&0 \\ 0&3&0 \\ 0&0&2 \end{bmatrix}, \begin{bmatrix} 2&0&0 \\ 0&2&0 \\ 0&0&2 \end{bmatrix}, \begin{bmatrix} 2&0&0 \\ 0&2&0 \\ 0&0&2 \end{bmatrix}, \begin{bmatrix} 2&0&0 \\ 0&1&0 \\ 0&0&1 \end{bmatrix}, \begin{bmatrix} 2&0&0 \\ 0&0&0 \\ 0&0&1 \end{bmatrix},$$
$$\begin{bmatrix} 1&0&0 \\ 0&2&0 \\ 0&0&0 \end{bmatrix}, \begin{bmatrix} 0&0&0 \\ 0&2&0 \\ 0&0&1 \end{bmatrix}, \begin{bmatrix} 0&0&0 \\ 0&2&0 \\ 0&0&2 \end{bmatrix}, \begin{bmatrix} 1&0&0 \\ 0&2&0 \\ 0&0&2 \end{bmatrix}, \begin{bmatrix} 3&0&0 \\ 0&2&0 \\ 0&0&4 \end{bmatrix}, \begin{bmatrix} 3&1&0 \\ 0&4&1 \\ 0&0&3 \end{bmatrix}.$$

Вычислим матрицу средних значений λ_{ij}:

$$\begin{bmatrix} 1{,}67 & 0{,}83 & 0{,}08 \\ 0 & 2{,}08 & 0{,}17 \\ 0{,}08 & 0 & 1{,}83 \end{bmatrix}.$$

Снова воспользуемся критерием χ^2 для каждого λ_{ij}. Получим следующую матрицу эмпирических значений χ^2:

$$\begin{bmatrix} 6{,}4 & 11 & 11{,}46 \\ 0 & 5{,}24 & 10 \\ 11{,}46 & 0 & 6{,}36 \end{bmatrix}.$$

Как видно, для заданного уровня значимости и количества степеней свободы ни для одного λ_{ij} эмпирическое значение χ^2 не превышает критическое. Следовательно, нет оснований отвергать гипотезу о неизменности интенсивности заявок λ_{ij}.

Эта система относится к системам без последействия, так как заявки поступают независимо друг от друга и момент поступления очередной заявки не зависит от поступления предыдущих заявок. По классификации дисциплин буферизации заявок во входном потоке, данный поток относится к бесприоритетным потокам. С точки зрения классификации дисциплин обслуживания имеем приоритетную систему одиночного типа, так как всякий раз на обслуживание назначается только одна задача. Следовательно, согласно обозначениям Кендалла получаем систему $\overline{M_H} / \overline{G_H} / N$. Характеристики по каждому классу заявок $j = 1, ..., H$ идентичны СМО с однородным потоком. Подробное описание характеристик можно найти в [Алиев, 2009; Клейнрок, 1979; Willing, 1999].

Итак, определено, к какому типу систем массового обслуживания относится моделируемая система, и выявлены ее основные характеристики. Далее перейдем к постановке оптимизационной задачи для данной системы.

Оптимизация системы массового обслуживания «Разработчик»

Задачу оптимизации для данной системы можно сформулировать следующим образом: уменьшить время простоя разработчика, уменьшить время доработки. Ввиду использования в процессе разработки методологии экстремального программирования[1] долгосрочное планирование зачастую оказывается бесполезным. В данном случае наиболее подходящей является постановка задачи о назначениях.

Пусть $x_1, x_2, ..., x_n$ – разработчики, $t_1, t_2, ..., t_m$ — типы заданий, которые выполняются разработчиками, $f(x_i, t_j)$ – время выполнения *i*-м разработчиком *j*-го задания, $i = 1, ..., n;\ j = 1, ..., m$. Тогда необходимо найти допустимое назначение, у которого суммарное время выполнения работ будет минимальным. Наиболее эффективным методом решения данной задачи является «венгерский» алгоритм [Гинзбург, 2003]. Как известно, временная сложность оригинального алгоритма — $O(n^4)$. Однако лучший показатель временной сложности данного алгоритма – $O(n^3)$. Первый такой алгоритм описан в статье Диника и Кронрода (1969). Затем в начале 1970-х Эдмондс и Карп, а также Томидзава показали, что временная сложность может быть уменьшена до $O(n^3)$. В 1980-х были также представлены компьютерные реализации алгоритма (Буркард и Деригс, Джонкер и Волгенант, Карпането и др.) [Burkard, 2009]. Отличие приведенного ниже алгоритма заключается в том, что он позволяет решать задачи для прямоугольных матриц времени выполнения работ. В других модификациях, да и в оригинальном алгоритме рассматривается только квадратная матрица времени выполнения работ. Далее приведем решение поставленной задачи с использованием модифицированного «венгерского» алгоритма.

Сначала рассмотрим простой случай с ***n*** разработчиками, ***m*** задачами и значениями ***n*** = ***m***. Построим исходную матрицу времени выполнения работ (таблица 2).

Таблица 2. Время выполнения работ. $C_{ij}, i = 1, ..., n;\ j = 1, ..., m$ – время выполнения *i*-м сотрудником *j*-го типа задания.

	1	2	3	…	m
1	C_{11}	C_{12}	C_{13}	$C_{1...}$	C_{1m}
2	C_{21}	C_{22}	C_{23}	$C_{2...}$	C_{2m}
…	$C_{...1}$	$C_{...2}$	$C_{...3}$		$C_{...m}$
N	C_{n1}	C_{n2}	C_{n3}	$C_{n...}$	C_{nm}

Так как в качестве решения необходимо знать, какой разработчик должен выполнять ту или иную работу, то решение можно записать в виде массива *D*:

$$D = \{d_1, d_2, ..., d_n\}. \quad (1.1)$$

Здесь индекс *d* соответствует номеру работы, а значение d_j соответствует номеру разработчика, назначенного на *j*-ю работу. На каждом шаге алгоритма будем выбирать для каждой строки минимальное значение. Затем выберем наименьшее значение из минимумов по всем строкам. Если минимальные значения в разных строках совпадают, то берем значения, находящиеся в северо-западном углу матрицы (не всегда верно для матриц 2*2). Записываем номер

[1] Экстремальное программирование (Extreme Programming, XP) — дисциплина разработки программного обеспечения и ведения бизнеса в области создания программных продуктов, которая фокусирует усилия обеих сторон (программистов и бизнесменов) на общих целях [Бек К., 2010].

строки минимального значения в соответствующую ячейку массива *D*. Удаляем строку и столбец, на пересечении которых находится найденный элемент. Повторяем шаг с новой матрицей.

Преимущество данного алгоритма над стандартным «венгерским» заключается в том, что на каждом шаге вычислительная сложность уменьшается, так как снижается размерность исходной матрицы, соответственно уменьшается количество элементов, подверженных обработке. Однако и данный алгоритм можно оптимизировать.

На первом шаге алгоритма для каждой строки матрицы можно выбрать минимальное значение и номер столбца, в котором находится выбранный элемент (на каждом шаге алгоритма значения элементов матрицы не меняются). Учтем замечания о выборе минимального элемента для матриц размерности 2*2:

Шаг 1.

$$D = \{\underbrace{0, 0, \ldots, 0}_{m}\}, \tag{1.2}$$

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ & \cdots & & \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{bmatrix} \rightarrow M = \{M_1, M_2, \ldots, M_n\}, \tag{1.3}$$

$$M_i = \left\{\min(c_{i1}, c_{i2}, \ldots, c_{im}) = \overline{m}, J = \{j : c_{ij} = \overline{m}\}\right\}. \tag{1.4}$$

Каждый массив M_i содержит минимальное значение и индексы столбцов, в которых они встречаются. Дальнейшая работа ведется с массивом *M*. Чтобы не зависеть от порядка следования элементов в массиве *M* и иметь возможность удалять элементы из массива, модифицируем структуру элемента массива (1.4). Добавим номер строки:

$$M_i = \left\{i, \min(c_{i1}, c_{i2}, \ldots, c_{im}) = \overline{m}, J = \{j : c_{ij} = \overline{m}\}\right\}. \tag{1.5}$$

Шаг 2.

Здесь и далее по тексту под элементом массива *M* будем понимать вложенный массив, который описывается формулой (1.5). Находим M_i с наименьшим $\overline{m}$ и наименьшим значением *i*. Если элементов в массиве больше двух, то заносим *i* в массив *D* в ячейку с номером, совпадающим с наименьшим значением *j,* и обозначаем номер ячейки *j**. Затем M_i можно удалить из массива *M*, а все элементы со значением *j,* равным *j*,* заменить на элемент с минимальным значением, полученный из столбцов матрицы с другими номерами. Перенумеровать оставшиеся элементы массива *M* с нуля и вернуться на начало шага 2.

На последней итерации, когда в массиве *M* останется два элемента, необходимо сделать проверку на пересечение значений *j* в оставшихся элементах и выбрать непересекающиеся.

Теперь рассмотрим случай $\boldsymbol{n > m}$. Шаги 1 и 2 будут такими же, как и для равных значений. Отличие только в финальном шаге. На последней итерации в массиве *M* останется $\boldsymbol{n - m + 2}$ элемента. На основе данных предыдущих итераций об обработанных *i* и *j* получаем итоговую матрицу размерности $\boldsymbol{(n - m + 2)*2}$. Для каждой строки матрицы вычислим сумму первого элемента со вторыми элементами остальных строк. Индексы строк элементов, давших наименьшую сумму, заносятся в соответствующие позиции итогового массива *D*.

В случае $\boldsymbol{m > n}$ алгоритм будет применен $\frac{m}{n} + 1$ раз. Первый шаг такой же, как и ранее. На втором шаге вычеркиваем самый левый столбец матрицы, в котором содержится наименьший элемент. Строку, в которой находился этот элемент, помечаем временно использованной. Далее выбираем минимальный элемент по оставшимся строкам матрицы. Возвращаемся на начало шага 2 и удаляем очередной столбец матрицы. Если останется только одна непомеченная как временно используемая строка матрицы, то выбираем в этой строке минимальный элемент *m**.

Удаляем из матрицы столбец с наименьшим номером, который содержит *m**. После этого снимаем пометки о временном использовании на строках и возвращаемся к началу шага 2. Данная итерация продолжается, пока не будут обработаны все столбцы матрицы.

Аналогично первым двум случаям исходную матрицу можно представить в виде массива:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ & \cdots & & \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{bmatrix} \rightarrow M = \{M_1, M_2, \ldots, M_n\}. \quad (1.6)$$

В отличие от рассмотренных ранее алгоритмов, структура множества M_i будет иной:

$$M_i = \left\{ i, u = \overline{0,1}, \min(c_{i1}, c_{i2}, \ldots, c_{im}) = \overline{m}, J = \{ j : c_{ij} = \overline{m} \} \right\}. \quad (1.7)$$

Здесь параметр *u* принимает значение 0, если множество M_i разрешено использовать, и 1, если M_i временно использовано.

Шаг 2 данного алгоритма отличается от рассмотренных для случаев ***n = m, n > m*** тем, что множество M_i не удаляется из массива, а помечается как временно использованное. Данная пометка снимается со всех множеств, как только не найдено более свободных.

Таким образом, описан алгоритм для разового нахождения оптимального назначения. Однако более реалистичной является ситуация, когда разработку ведет группа специалистов и сама задача делится на подзадачи. Необходимо подобрать группу специалистов для максимально быстрого решения общей задачи. Иными словами, необходимо получить оптимальную последовательность задач о назначении.

Сначала рассмотрим случай, когда части основной задачи не зависят друг от друга и могут быть обработаны в любом порядке. Пусть имеется *N* разработчиков и *M* типов задач, и основная задача распадается на *K* подзадач:

$$T = \{C_k, k = \overline{1,K}\}, \quad (1.8)$$

где C_k представляют собой матрицы назначения.

Рассмотрим ситуацию $N = M = K = 3$. Тогда получаем три матрицы:

$$C_1 = \begin{bmatrix} c_{11}^1 & c_{12}^1 & c_{13}^1 \\ c_{21}^1 & c_{22}^1 & c_{23}^1 \\ c_{31}^1 & c_{32}^1 & c_{33}^1 \end{bmatrix} C_2 = \begin{bmatrix} c_{11}^2 & c_{12}^2 & c_{13}^2 \\ c_{21}^2 & c_{22}^2 & c_{23}^2 \\ c_{31}^2 & c_{32}^2 & c_{33}^2 \end{bmatrix} C_3 = \begin{bmatrix} c_{11}^3 & c_{12}^3 & c_{13}^3 \\ c_{21}^3 & c_{22}^3 & c_{23}^3 \\ c_{31}^3 & c_{32}^3 & c_{33}^3 \end{bmatrix}. \quad (1.9)$$

Применив первый шаг алгоритма, получим три массива для каждой из матриц:

$$M^1 = \{M_1^1, M_2^1, M_3^1\} \; M^2 = \{M_1^2, M_2^2, M_3^2\} \; M^3 = \{M_1^3, M_2^3, M_3^3\}. \quad (1.10)$$

Применяя алгоритм, получаем три массива решений:

$$D^1 = \{d_1^1, d_2^1, d_3^1\} \; D^2 = \{d_1^2, d_2^2, d_3^2\} \; D^3 = \{d_1^3, d_2^3, d_3^3\}. \quad (1.11)$$

Полученные результаты можно записать в виде матрицы. Такая запись даст возможность управлять последовательностью работ путем перестановок отдельных строк матрицы.

Таким образом, получено решение задачи о назначениях для задач сложного типа без зависимостей между подзадачами. Был рассмотрен случай для трех подзадач, но данный алгоритм действителен и для любого целого положительного K.

Далее усложним задачу и введем зависимости между подзадачами. Для описания последовательности работ построим орграф, вершинами которого являются подзадачи:

$$G = \{g_k, k = \overline{1,K}\}. \quad (1.12)$$

Пусть каждая вершина имеет вес, равный суммарному времени выполнения работ по подзадаче. Тогда решением задачи о назначениях с зависимыми подзадачами станет нахождение пути проходящего через все вершины графа. Суммарное время выполнения работ – сумма весов вершин графа.

В таблице 3 приведены результаты практической проверки времени работы алгоритма.

Таблица 3. Время работы алгоритма (в секундах) в зависимости от размерности матрицы. Каждое соотношение проверялось по 10 раз

	10	20	30	40	50	60	70	80	90	1600
1	0,016	0,004	0,008	0,005	0,013	0,018	0,007	0,008	0,014	9,327
2	0,004	0,004	0,005	0,005	0,006	0,006	0,007	0,009	0,009	9,442
3	0,010	0,004	0,004	0,007	0,005	0,006	0,007	0,007	0,005	9,426
4	0,004	0,004	0,11	0,005	0,012	0,008	0,012	0,014	0,018	9,282
5	0,003	0,004	0,004	0,005	0,006	0,011	0,007	0,011	0,012	9,412
6	0,005	0,004	0,005	0,005	0,012	0,012	0,009	0,010	0,019	9,434
7	0,005	0,004	0,006	0,004	0,005	0,006	0,016	0,015	0,015	9,382
8	0,004	0,004	0,005	0,005	0,010	0,007	0,009	0,012	0,009	9,533
9	0,004	0,006	0,005	0,011	0,005	0,006	0,010	0,008	0,017	9,541
10	0,004	0,006	0,005	0,005	0,006	0,006	0,008	0,008	0,013	9,714
Среднее	0,006	0,004	0,015	0,006	0,008	0,009	0,009	0,010	0,013	9,449

Алгоритм реализован на языке С, временная сложность алгоритма $O(n^3)$. Тесты проводились на ноутбуке Acer Aspire 5100. Следует также заметить, что значительную роль на увеличение времени обработки с увеличением размерности входной матрицы играет вывод результата. В таблице 3 приведено суммарное время работы (расчет и вывод).

Для примера работы алгоритма рассмотрим матрицу 4*4:

$$\begin{bmatrix} 1\,2\,3\,4 \\ 4\,3\,2\,1 \\ 1\,2\,3\,4 \\ 4\,3\,3\,4 \end{bmatrix}.$$

На первом шаге получаем массив *M*, содержащий четыре элемента (множества) M_0, M_1, M_2, M_3:

$$M_0 = \{i = 0, \min = 1, J = [0]\};$$
$$M_1 = \{i = 1, \min = 1, J = [3]\};$$
$$M_2 = \{i = 2, \min = 1, J = [0]\};$$
$$M_3 = \{i = 3, \min = 3, J = [1, 2]\}.$$

Выбираем элемент M_0 массива *M* (так как минимальное значение по строке совпадает в трех строках матрицы *M*, то выбираем элемент с наименьшим номером строки *i*), результат заносим в массив *D* ($d_0 = j$). Далее работаем с матрицей 3*3, которую теперь описывает массив *M*, состоящий из трех элементов (элементы массива определяются по оставшимся строкам и столбцам матрицы *M* и нумеруются с нуля):

$$M_0 = \{i = 1, \min = 1, J = [3]\};$$
$$M_1 = \{i = 2, \min = 2, J = [1]\};$$
$$M_2 = \{i = 3, \min = 3, J = [1, 2]\}.$$

Так как M_0 снова имеет минимальное значение, то заносим его в массив *D* и продолжаем работать с матрицей 2*2. Строим новый массив *M* (элементы массива определяются по оставшимся строкам и столбцам матрицы *M* и нумеруются с нуля):

$$M_0 = \{i = 2, \min = 2, J = [1]\};$$
$$M_1 = \{i = 3, \min = 3, J = [1, 2]\}.$$

Результаты обработки данного массива заносятся в массив $D = \{0, 2, 3, 1\}$. Таким образом, после данной итерации результирующий массив *D* содержит следующую информацию:

для выполнения задания 0 требуется сотрудник 0;
для выполнения задания 1 требуется сотрудник 2;
для выполнения задания 2 требуется сотрудник 3;
для выполнения задания 3 требуется сотрудник 1.

Итак, описана задача оптимизации, возникающая в подсистеме «Разработчик», и приведен метод ее решения.

Однако данный метод позволяет решать задачу статически, то есть ничего не известно о поведении данной системы во времени. Еще одним недостатком данного метода является невозможность определить взаимосвязь таких важных факторов, как время выполнения работ, сложность работы и стоимость исполнения работы. Добавление всех этих параметров к рассмотренной задаче увеличит в разы ее размерность, следовательно, и время расчетов. Время расчетов может быть настолько велико, что не уложится в приемлемые рамки. Решение такой задачи требует применения совершенно иных методов.

Имитационное моделирование подсистемы «Разработчик»

Одним из таких методов является имитационное моделирование, известными достоинствами которого являются гибкость и возможность моделирования систем на любом уровне детализации или абстракции. Первоначально имитационное моделирование определялось как дискретно-событийный способ описания и моделирования процессов, присущих системам массового обслуживания, системам с отказами и восстановлением элементов, дискретными производствами и т. д. [Бахвалов, 1997; Угольницкий, Тихонов, 2011; Шрайбер, 1980] .

Наиболее подходящим инструментом для решения поставленной ранее задачи (распределение ресурсов во времени, описание связей различных факторов системы), является агентное моделирование[2].

Агентное моделирование является инструментом, при помощи которого возможно успешное моделирование сложных адаптивных систем. Агентное моделирование базируется на идее моделирования процессов «сверху-вниз»: в основе модели лежит набор основных элементов, из взаимодействия которых рождается обобщенное поведение системы. Агентами могут быть не только индивидуумы, но и группы индивидуумов, подсистемы и т. д. Применение данного подхода к моделированию наиболее удобно в случаях, когда представляют интерес характеристики поведения системы в целом, которые определяются как интегральные характеристики всей совокупности агентов [Борщев, 2004].

Данная модель позволит объединить решения, полученные на предыдущих этапах, для получения имитационной модели, максимально приближенной к реальности. С использованием формул, описывающих характеристики системы массового обслуживания, можно получить ограничения среды, в которой обитает агент. Решение задачи о назначениях, а также

[2] Агентное моделирование (agent-based modeling (ABM)) — метод имитационного моделирования, исследующий поведение децентрализованных агентов и то, как такое поведение определяет поведение всей системы в целом. При моделировании определяется поведение агентов на индивидуальном уровне, а глобальное поведение возникает как результат деятельности множества агентов.

модифицированный «венгерский» алгоритм использованы как составные части алгоритма, описывающего поведение агента.

Наиболее распространенным способом описания поведения агентов являются карты состояний (statecharts), которые представляют собой конечные автоматы с некоторыми дополнениями. Они описывают переходы между состояниями и события, инициирующие данные переходы, временные задержки и действия, совершаемые агентом на протяжении своей жизни. Агент может иметь несколько параллельно активных и взаимодействующих карт состояний, каждая из которых описывает какой-либо аспект его жизни [Arthur, 1991; Macal, North, 2005].

Для рассматриваемой подсистемы можно выделить следующих агентов: заявки на доработки, разработчики.

Простая схема работы заключается в том, что заявки попадают в систему, обслуживаются, задерживаются и покидают систему. Событие генерации очередной заявки будет соответствовать созданию нового агента. На рисунке 1 изображена схема жизненного цикла агента заявки.

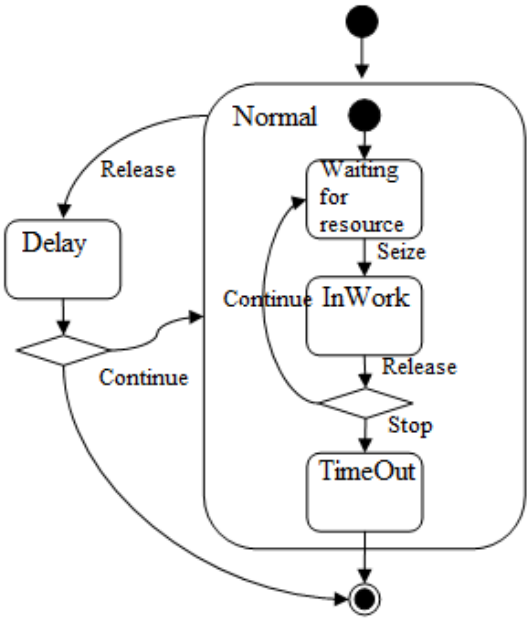

Рис. 1. Схема жизненного цикла агента заявки (см. пояснения в тексте)

Для каждой заявки можно выделить два основных состояния: в работе (на диаграмм — Normal) и отложено (на диаграмме — Delay).

Как только появляется новая заявка, она дожидается появления свободного ресурса для ее обработки. При появлении необходимого объема ресурсов заявка поступает в обработку. После обработки занятые ресурсы освобождаются, и заявка переходит на последний этап — верификация выполнения. В результате верификации определяется степень выполнения заявки и выявляется необходимость каких-либо доработок. В случае, когда необходимы доработки, заявка снова попадает в состояние ожидания свободных ресурсов. Из этого состояния заявка либо возвращается к рабочему циклу, либо завершается вовсе.

Поведение агентов, обрабатывающих заявки, можно описать тремя основными состояниями: свободен (Idle), занят (Busy), перерыв (TimeOut). Схема жизненного цикла агента обработчика заявок приведена на рисунке 2.

При появлении заявки обработчик переходит в состояние «занят» и выполняет работу. После завершения работ обработчик попадает в состояние «перерыв», так как моментально приступить к обработке новой заявки он не может.

Таким образом, завершен последний этап моделирования подсистемы «Разработчик». При помощи программного комплекса AnyLogic (данный ПК предоставляет возможности для

разработки моделей различных классов) были проведены имитационные эксперименты. Результаты имитационных экспериментов, проведенных с использованием построенной агентной модели, показали неэффективность существовавшего процесса разработки. Были даны рекомендации по разделению разработчиков на группы, специализирующиеся по видам деятельности: специализация на отчетах, клиентских интерфейсах и бизнес-логике. Помимо непосредственного процесса кодирования был взят в рассмотрение процесс проектирования как отдельный тип задач, соответственно проектировщики также рассматриваются как обслуживающий прибор.

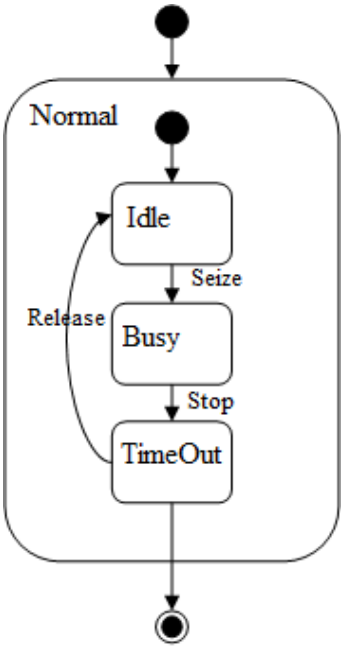

Рис. 2. Схема жизненного цикла агента обработчика заявок (см. пояснения в тексте)

Ранее эффективность оценивалась с использованием жестких планов: определенное число задач должно быть выполнено за некоторое время. Этот подход имел массу недостатков, в нем не учитывалась сложность задач и многие другие аспекты. Использование имитационных экспериментов позволило ввести гибкий график работ, предварительно рассчитывать необходимое для конкретной задачи время работы над ней и соответственно снизить количество часов переработок. Это напрямую отразилось на качестве продукта и моральном состоянии коллектива. При этом производительность труда увеличилась на десять процентов.

Список литературы

Алиев Т. И. Основы моделирования дискретных систем // СПб.: СПбГУ ИТМО. — 2009. — С. 77–165.

Бахвалов Л. Компьютерное моделирование — длинный путь к сияющим вершинам // Компьютерра. — 1997. — № 40 (217). — С. 26–36.

Бек К. Экстремальное программирование // СПб.: Питер. — 2010.

Борщев А. В. Практическое агентное моделирование и его место в арсенале аналитика // Exponenta Pro. — 2004. — № 3–4. — С. 38–47.

Гинзбург А. И. Экономический анализ: Предмет и методы. Моделирование ситуаций. Оценка управленческих решений: учебное пособие // СПб.: Питер. — 2003. — 622 с.

Джейсоул Н. К. Очереди с приоритетами / Пер с англ. М.: Мир. — 1973. — 279 с.

Елиферов В. Г. Управление качеством. Сказки, мифы и проза жизни // М.: Вершина. — 2006. — 296 с.

Карпов Ю. Г. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5 / СПб.: БХВ-Петербург. — 2005. — 400 с.

Клейнрок Л. Теория массового обслуживания / Пер. с англ. Грушко И.И.; ред. Нейман В.И. — М.: Машиностроение. — 1979. — 432 с.

Угольницкий Г., Тихонов С. Имитационное моделирование бизнес-процессов: системы массового обслуживания / LAP Lambert Academic Publishing. — 2011. — 166 с.

Шрайбер Т. Дж. Моделирование на GPSS / Пер. с англ. Гаргера В. Л. , Шмуйлович И. Л.; ред. Файнберг М. А. / М.: Машиностроение. — 1980. — 592 с.

van der Aalst W. M. P., ter. Hofstede A. H. M. YAWL: Yet another workflow language (revised version) // Queensland Univ. Technol., Brisbane, Australia: QUT Tech. Rep. - FIT-TR-2003-04. — 2003.

Arthur, W. B. Designing Economic Agents that Act Like Human Agents: A Behavioral Approach to Bounded Rationality // The American Economic Review. — 1991. — № 81 (2). — P. 353–359.

Burkard, R. E. Assignment problems/Rainer Burkhard, Mauro Dell'Amico, Silvano Martello // ISBN 978-0-898716-63-4. — Vol. 4. — P. 77.

Havey M. Essential business process modeling. — Sebastopol, CA: O'Reilly, 2005.

Hofacker I., Vetschera R. Algorithmical approaches to business process design // Comp. Oper. Res. — 2001. — Vol. 28. — P. 1253–1275.

Koubarakis M., Plexousakis D. A formal framework for business process modeling and design // Inf. Syst. — 2002. — Vol. 27. — P. 299–319.

Macal C., North M. Tutorial on agent based modeling and simulation / Proceedings of the 2005 Winter Simulation Conference, Center for Complex Adaptive Systems Simulations (CAS) // Argonne National Laboratory.

Phalp K., Shepperd M. Quantitative analysis of static models of processes // Syst. Software – 2000. — Vol. 52. — P. 105–112.

Vergidis K. et al. Business Process Analysis and Optimization: Beyond Reengineering // IEEE Transactions on Systems, Man, and Cybernetics. — 2008. — Part C. — Vol. 38, No. 1.

Willing A. A Short Introduction to Queuing Theory // Technical University Berlin. — July 21, 1999.

White S. Business Process Modeling Notation (BPMN): specification. — 2004. — Version 1.0. — May 3.