ISSN 1684–8853

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ

НАУЧНО-ПРАКТИЧЕСКИЙ ЖУРНАЛ

5(18)/2005

НАУЧНО-ПРАКТИЧЕСКИЙ ЖУРНАЛ

5(18)/2005

РЕЦЕНЗИРУЕМОЕ ИЗДАНИЕ

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ

Адрес редакции: 190000, Санкт-Петербург,
Б. Морская ул., д.67
Тел.: (812) 710-66-42, (812) 313-70-88
Факс: (812) 313-70-18
E-mail: ius@aanet.ru
Сайт: www.i-us.ru

Журнал зарегистрирован
в Министерстве РФ по делам печати,
телерадиовещания и средств массовых коммуникаций.
Свидетельство о регистрации ПИ № 77-12412 от 19 апреля 2002 г.

Журнал распространяется по подписке.
Подписку можно оформить через редакцию, а также
в любом отделении связи по каталогам:
«Пресса России» – № 42476;
агентства«Роспечать»:
«Газеты и журналы» – № 15385,
«Издания органов НТИ» – № 69291.

ОБРАБОТКА ИНФОРМАЦИИ И УПРАВЛЕНИЕ

Муромцев Д. Ю. *Информационная система энергосберегающего управления сложными объектами* 2

Сабонис С. С. *Алгоритмы диагностирования автоматизированной системы контроля уровня воды* 6

МОДЕЛИРОВАНИЕ СИСТЕМ И ПРОЦЕССОВ

Зикратов И. А. *Метод автоматического отождествления линий равных высот при создании цифровых карт местности на основе картометрического подхода* 11

ПРОГРАММНЫЕ И АППАРАТНЫЕ СРЕДСТВА

Колесов Н. В., Толмачева М. В. *Составление расписаний решения задач в конвейерных вычислительных системах* 16

Наумов Л. А. *Решение задач с помощью клеточных автоматов посредством программного обеспечения CAME&L (Часть I)* 22

ИНФОРМАЦИОННЫЕ КАНАЛЫ И СРЕДЫ

Рыжиков Ю. И. *Расчет многоуровневой системы очередей* 31

СИСТЕМНЫЙ АНАЛИЗ

Перовская Е. И. *Системный анализ и имитационное моделирование как средство объединения результатов гуманитарных и точных наук* 35

Колотаев А. В. *Реализация библиотеки имитационных моделей как набора обобщенных компонент* 47

ХРОНИКА И ИНФОРМАЦИЯ

Шалыто А. А. *Никлаус Вирт – почетный доктор Санкт-Петербургского государственного университета информационных технологий, механики и оптики* 56

Памяти Перовской Евгении Ивановны 59

СВЕДЕНИЯ ОБ АВТОРАХ

60

АННОТАЦИИ

62

ЛР № 010292 от 18.08.98.
Сдано в набор 20.08.2005. Подписано в печать 01.10.2005. Формат 60×901/8.
Бумага офсетная. Гарнитура SchoolBookC. Печать офсетная.
Усл. печ. л. 8,0. Уч.-изд. л. 9,0. Тираж 1000 экз. Заказ 422.

Оригинал-макет изготовлен
в отделе электронных публикаций и библиографии ГУАП.
190000, Санкт-Петербург, Б. Морская ул., 67.

Отпечатано с готовых диапозитивов
в отделе оперативной полиграфии ГУАП.
190000, Санкт-Петербург, Б. Морская ул., 67.

УДК 638.512.011.56.001.57.681.5

ИНФОРМАЦИОННАЯ СИСТЕМА ЭНЕРГОСБЕРЕГАЮЩЕГО УПРАВЛЕНИЯ СЛОЖНЫМИ ОБЪЕКТАМИ

Д. Ю. Муромцев,
канд. техн. наук, доцент
Тамбовский государственный технический университет

Рассматривается комплекс задач анализа и синтеза оптимального энергосберегающего управления типовыми динамическими объектами различных классов с учетом возможных изменений состояний функционирования при эксплуатации. Для оперативного решения задач анализа и синтеза оптимального управления используется комбинация методов принципа максимума, динамического программирования и синтезирующих переменных. Приводятся алгоритмы синтеза оптимальных управляющих воздействий в реальном масштабе времени.

The set of tasks aimed at analysis and synthesis of optimum energy saving control by standard dynamic object of different types with regard for possible changes of functioning states in operation is considered. The combination of methods of maximum principle, dynamic programming and synthesizing variables is applied for operative solution of tasks of analysis and synthesis of optimum control. The algorithms of synthesis of optimum controlled effect in real time scope are given.

Во многих отраслях промышленности, в том числе машиностроительной, электротехнической, металлургической, строительных материалов, широко используются аппараты с электронагревом. Энергетические затраты на ведение технологических процессов в этих аппаратах составляют значительную долю в себестоимости продукции. С учетом роста цен на электроэнергию эти затраты становятся сопоставимыми с затратами на сырье.

Важным резервом снижения энергозатрат в тепловых аппаратах и машинах с электроприводами является оптимальное управление динамическими режимами. Теоретические и экспериментальные исследования показывают, что при оптимальном управлении (ОУ) отдельными аппаратами снижение затрат энергии в динамических режимах может составлять 10–25 % по сравнению с традиционным управлением. Дополнительный эффект достигается, если система управления группой аппаратов обладает интеллектуальными свойствами по использованию базы знаний и сложившейся производственной ситуации для выбора наилучшего варианта закрепления обрабатываемых деталей за определенными аппаратами.

Разрабатываемая интеллектуальная информационная система позволяет решать задачи энергосберегающего управления применительно к двум классам сложных объектов. К первому классу относятся объекты со многими входами и многими выходами, т. е. типа MIMO-систем (Multi Input Multi Output), например, многозонные электрические печи. Главной особенностью этих объектов является то, что каждый вход влияет на несколько выходов.

Ко второму классу относятся группы объектов с сосредоточенными параметрами, например, участок термообработки с несколькими электрическими камерными печами. Здесь задаче управления динамическим режимом предшествует задача распределения термообрабатываемых изделий между отдельными печами. Этот класс объектов условно обозначим MSISO (Multi Signal Input Signal Output).

Общей особенностью рассматриваемых объектов является нелинейность модели динамики, вызываемая широким температурным диапазоном (до 1000 °С и более). Поэтому для решения задач энергосберегающего управления приходится использовать многостадийные модели в виде дифференциальных уравнений с разрывной правой частью [1, 2].

Введем следующие обозначения, отражающие особенности рассматриваемых классов объектов: m – число входов (и, соответственно, выходов) объекта MIMO, применительно к многозонной

печи m равно числу зон; k – число стадий модели с одним входом; n – число отдельных объектов в MSISO; d – число видов деталей для термообработки; l – пространственная координата в объекте с распределенными параметрами. Этим обозначениям соответствуют множества M, K, N, D и L.

С учетом рассмотренных особенностей выделим следующие виды задач оптимального управления (ЗОУ) объектами MIMO и MSISO.

1. Элементарная ЗОУ, или Э-задача, в которой $m = 1$ (или $n = 1$), $k = 1$ и $d = 1$.

2. К-задача, для которой $m = 1$ (или $n = 1$), $k \geq 2$ (и $d = 1$).

3. М-задача, для которой $m \geq 2$ и $k = 1$.

4. МК-задача с $m \geq 2$, $k \geq 2$.

5. ML и MKL – задачи для MIMO, в которых учитываются температуры и скорости их изменения по длине печи.

6. ND-задача для MSISO по выбору варианта закрепления термообрабатываемых деталей за печами.

В литературе по оптимальному управлению достаточно освещены вопросы решения Э-, К- и М-задач [3–5]. Остановимся подробнее на МК-, ML-, MKL- и ND-задачах.

МК-задача формулируется следующим образом. Объект, динамика которого описывается моделью

$$\dot{z} = A\left(m,\ k_j\right)z(t) + B\left(m,\ k_j\right)u(t),\quad j = \overline{1,\ m};$$

$$z \overset{\Delta}{=} \left(z_1,\ \ldots,\ z_m\right)^T;\quad u \overset{\Delta}{=} \left(u_1,\ \ldots,\ u_m\right)^T, \qquad (1)$$

требуется перевести из начального состояния z^0 в конечное $z^к$ за фиксированное время, т. е.

$$z\left(t_0\right) = z^0,\quad z\left(t^к\right) = z^к \qquad (2)$$

при ограничении на управление

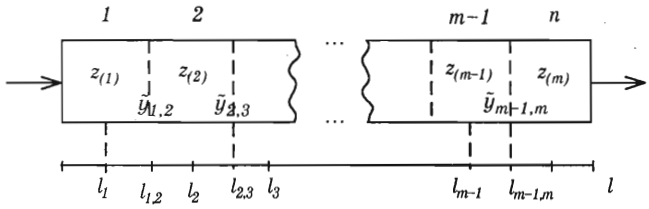
$$\forall t \in \left[t_0,\ t_к\right]:\quad u_i(t) \in \left[u_н,\ u_в\right],\quad i = \overline{1,m} \qquad (3)$$

и минимизируемом функционале

$$J = \int_{t_0}^{t_к} \left(z^T Q_z z(t) + u^T Q_u u(t)\right)dt, \qquad (4)$$

где $A\left(m,\ k_j\right)$, $B\left(m,\ k_j\right)$ – матрицы параметров блочной структуры, отражающей число стадий k_j – для каждой зоны; $u_н$, $u_в$ – границы изменения компонентов вектора управления u; Q_z, Q_u – матрицы весовых коэффициентов.

В ML- и MKL- задачах дополнительно к фазовым координатам $z_{(1)}$, ..., $z_{(m)}$, характеризующим температурные режимы в центральных частях зон, задается информация о максимальных скоростях изменения температуры y по длине l печи. Результаты имитационных экспериментов показывают, что эти изменения приходятся на точки,



■ *Рис. 1. Температурные режимы по длине печи*

соответствующие межзонным переходам $l_{j,j+1}$, $j = \overline{1,m-1}$ (рис. 1).

Кроме того, значения $y(l)$ в окрестности точек $l_{j,j+1}$ с достаточной точностью могут быть описаны функциональными зависимостями от значений температур в центрах соседних зон, т. е. $l \in [y_{j,j+1} - \Delta l,\ y_{j,j+1} + \Delta l\ ]$: $y(l) = f_j(y_j; y_{j+1})$.

ND-задача в общем случае формулируется следующим образом. Задаются модели динамики для каждого объекта, т. е. множество моделей

$$\left\{\dot{z}_{(i)} = f_{ij}\left(z_{(i)},\ u_{(i)}t;\ A_{ij},\ B_{ij}\right),\quad i = \overline{1,\ m}\right\}; \qquad (5)$$

а также ограничения на управление в каждый момент времени

$$\left\{\forall t \in T_i:\quad u_i(t) \in \left[u_{iн},\ u_{iв}\right],\quad i = \overline{1,m}\right\} \qquad (6)$$

и общее (на лимит используемой энергии)

$$\sum_{i=1}^{m} \int\limits_{t \in T_i} u_{(i)}(t)dt \leq J_л; \qquad (7)$$

двумерный массив исходных данных R_{ij}, которые требуются для решения задачи управления i-м объектом при обработке деталей j-го вида:

$$R = \left\{R_{ij};\ i = \overline{1,m};\quad i = \overline{1,d}\right\}; \qquad (8)$$

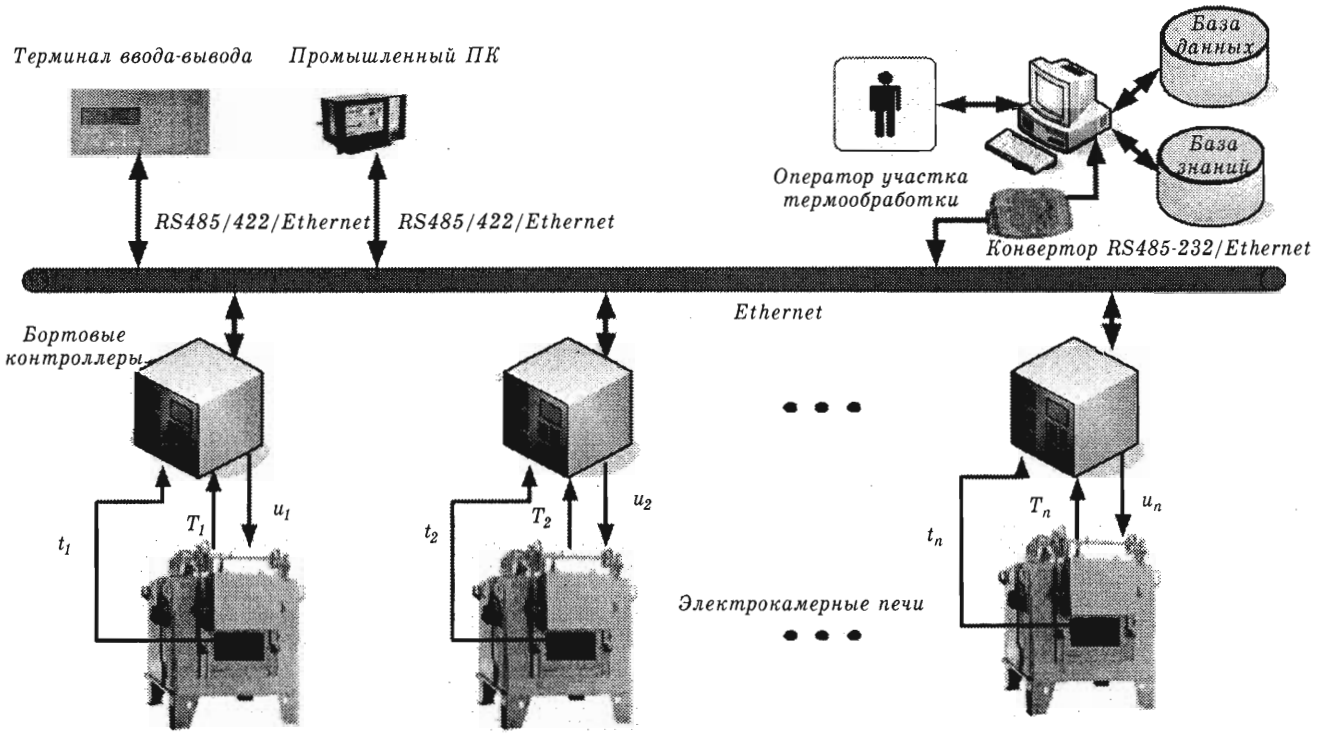
плановое задание выпуска продукции на временном интервале T

$$П = \left\{П_j,\ \ j = \overline{1,d}\right\}; \qquad (9)$$

результаты полного анализа ЗОУ для моделей динамики M_i, $i = \overline{1,\ m}$ (см. формулу (5)), возможных функционалов F, стратегий реализации управления S и ограничений (например, (6) и др.), т. е. четверок $K_i = <M, F, S, O>$, $i = \overline{1,m}$; $j = \overline{1,\ d}$; алгоритмы решения задачи распределения планового задания (9) между печами.

Требуется распределить плановое задание (9) между печами с учетом ограничений (6), (7). При этом общие затраты энергии должны быть минимальны:

$$J_э = \sum_{i=1}^{m} \int\limits_{t \in T} \left(u_i u_{(i)}\left(t;\ R_{ij}\right)(t)dt\right) \to \min_{u_{(i)},\ R_{ij}}. \qquad (10)$$



■ ***Рис. 2.*** *Схема системы управления группой печей ($t_1 \ldots t_n$ – температура внутри камеры печи; $T_1 \ldots T_n$ – температура снаружи камеры печи; $u_1 \ldots u_n$ – управляющие воздействия)*

В формулах (5)–(9) $z_{(i)}$, $u_{(i)}$ – вектор фазовых координат и управление для i-й печи; $f_{ij}(\,)$ – функция, определяющая правую часть дифференциального уравнения для i-й печи при выпуске j-го изделия; A_{ij}, B_{ij} – параметры модели динамики с функцией f_{ij}, $u_{i\text{н}}$, $u_{i\text{в}}$ – нижняя и верхняя границы изменения $u_{(i)}$; $J_{\text{л}}$ – выделенный лимит энергии на временной интервал T; $T_i \subseteq T$ – временной интервал работы i-й печи; Π_j – плановое задание по j-му виду продукции.

Результатом решения задачи распределения должны быть исходные данные R_{ij}^* для решения задач оптимального управления режимами работы печей. Задачи оптимального управления, т. е. Э- и К-задачи решаются бортовыми контроллерами с использованием базы знаний экспертной системы, содержащейся в промышленном компьютере [6].

Схема информационно-управляющей системы приведена на рис. 2.

Задача синтеза управления в реальном времени, решаемая бортовым контроллером, существенно облегчается, если вместо ОУ рассчитывается квазиоптимальное управление (КОУ). Проблема квазиоптимального управления связана в ряде случаев с невозможностью плавно изменять оптимальное управление по требуемому закону, а также со сложностью расчетов точного значения функции ОУ. Задачу синтеза квазиоптимального управления будем рассматривать как частный случай задачи синтеза ОУ.

В качестве основного вида КОУ $\tilde{u}(t)$ обычно рассматривается ступенчатая функция, которая с требуемой точностью или допустимым увеличением функционала аппроксимирует непрерывную функцию $u^*(t)$. Применение КОУ позволяет значительно упростить реализацию управляющих воздействий за счет небольшого числа фиксированных значений $u(t)$. Например, для электронагрева используется два нагревательных элемента, в этом случае $u(t)$ может принимать три значения: 0 (оба элемента выключены), u_1 (один элемент включен) и u_2 (два элемента включены). Число значений $\tilde{u}(t)$ может увеличиваться как за счет введения дополнительных элементов, так и за счет разных способов их включения (последовательное, параллельное, комбинированное).

Повышение точности управления при использовании КОУ достигается за счет увеличения числа «ступенек» (уровней). При этом возможны два случая. В первом случае задается m фиксированных уровней значений управляющего воздействия, т. е. $\tilde{c}_1$, $\tilde{c}_2$, ..., $\tilde{c}_m$, и требуется рассчитать моменты переключения $t_1, t_2, \ldots, t_{m-1}$. Во втором случае фиксируются моменты переключения и определяются значения управляющих воздействий. Задаваемое число «ступенек» m зависит, с одной стороны, от возможности технической реализации, например, числа ТЭНов, с другой стороны, от требуемой точности аппроксимации ОУ.

Эффективным КОУ $\tilde{u}^*(t)$ называется КОУ, которое обеспечивает увеличение функционала на некоторую достаточную величину ΔJ, % по сравнению с J^* при ОУ $u^*(t)$. Если не оговариваться особо, то берется $\Delta J = 1\%$. Обеспечение требуемой эффективности $\tilde{u}^*(t)$ достигается увеличением числа рассчитываемых параметров. Для их расчета может использоваться метод неопределенных множителей Лагранжа.

Рассмотренная информационная система внедряется на ФГУП "Тамбовский завод "Октябрь".

Литература

1. **Филиппов А. Ф.** Дифференциальные уравнения с разрывной правой частью. М.: Наука, 1985. – 224 с.
2. **Муромцев Ю. Л., Орлова Л. П., Муромцев Д. Ю.** Идентификация моделей, учитывающих изменение состояний функционирования // Обработка сигналов и полей. – 2000. – № 3. – С. 45–48.
3. **Муромцев Ю. Л., Ляпин Л. Н., Попова О. В.** Моделирование и оптимизация систем при изменении состояний функционирования. – Воронеж: Изд-во ВГУ, 1992. – 164 с.
4. **Муромцев Д. Ю., Муромцев Ю. Л., Орлова Л. П.** Синтез энергосберегающего управления многостадийными процессами комбинированным методом //Автоматика и телемеханика. – 2002. – № 3. – С. 169–178.
5. **Муромцев Д. Ю.** Оперативный синтез энергосберегающего управления для линейных систем с запаздыванием на множестве состояний функционирования // Тр. ТГТУ: Сб. науч. Ст. молодых ученых и студентов. Тамбов, 1999. – Вып. 4. – С. 47–50.
6. **Муромцев Д. Ю., Орлов В. В.** Информационно-технологическая среда проектирования интеллектуальных контроллеров // Компьютерная хроника. – 1997. – № 12. – С. 3–8.

ПАМЯТКА ДЛЯ АВТОРОВ

Поступающие в редакцию статьи проходят обязательное рецензирование.

При наличии положительной рецензии статья редактируется и рассматривается редакционной коллегией. Принятая в печать статья направляется автору для согласования редакторских правок. После согласования автор представляет в редакцию окончательный вариант текста статьи, а также фотографию и краткое изложение сведений о себе.

Процедуры согласования текста статьи, предоставления фото (размером 4×5,5 см) и сведений об авторе могут осуществляться как непосредственно в редакции, так и по e-mail (электронный вариант фото в виде файла *.tif, *.jpg с разрешением 300 dpi).

При отклонении статьи редакция представляет автору мотивированное заключение и рецензию. При необходимости доработать статью — рецензию.

Редакция журнала напоминает, что ответственность за подбор, достоверность и точность фактов, экономико-статистических и технических показателей, собственных имен и прочих сведений, а также за то, что в материалах не содержится сведений, не подлежащих открытой публикации, несут авторы публикуемых в журнале материалов и рекламодатели.

УДК 681.518.5

АЛГОРИТМЫ ДИАГНОСТИРОВАНИЯ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ КОНТРОЛЯ УРОВНЯ ВОДЫ

С. С. Сабонис,
аспирант,
Санкт-Петербургский государственный политехнический университет

В статье рассматривается автоматизированная система контроля уровня воды ОВУ-214 («Водоотливная станция»). Для диагностирования данной системы применяются алгоритмы обнаружения дефектов: модифицированный алгоритм кумулятивных сумм для обнаружения изменения среднего и алгоритм Гиршика–Рубина–Ширяева для обнаружения изменения среднего и дисперсии, в качестве модели диагностирования рассматривается модель аналогового датчика, измеряющего уровень воды в сборной емкости водоотливной станции.

The water level monitoring automated system is considered in this paper. The modified CUSUM algorithm and the algorithm by Girshik, Rubin and Shiryaev are applied to the system diagnose. The model of the analog sensor, measuring water level, is used as a model of diagnostic object.

Введение

В работе [1] проведен сравнительный анализ алгоритмов обнаружения разладки случайного процесса на примере процессов авторегрессии 1-го и 2-го порядков. Среди множества алгоритмов обнаружения выбраны оптимальные для двух типов дефектов: 1) изменение среднего; 2) изменение дисперсии.

При этом как критерий качества обнаружения используется среднее время обнаружения дефекта при контроле уровня вероятности ложного обнаружения.

Итак, среди набора алгоритмов обнаружения изменения среднего наилучшим является модифицированный алгоритм кумулятивных сумм (АКС-м) [2]. Среди набора алгоритмов обнаружения изменения среднего и дисперсии наилучшим является алгоритм Гиршика–Рубина–Ширяева (ГРШ) [3, 4].

Рассмотрим общую структуру системы диагностирования (рис. 1). На объект диагностирования действуют внешние факторы, что приводит к возникновению дефектов. Сигнал на выходе объекта диагностирования подается на блок, реализующий алгоритм принятия решений, который на выходе должен выдать сигнал о наличии или отсутствии дефекта. Алгоритм принятия решений состоит в формировании решающей статистики $G(n)$ на основе выходного сигнала объекта $z(n)$ и сравнении ее с порогом h.

Модифицированный алгоритм кумулятивных сумм:

$$G(n)=\max\left\{0,G(n-1)+\operatorname{sign}\left(z(n)-\mu_1-\upsilon/2\right)\right\},\ \mu_1<\mu_2;$$

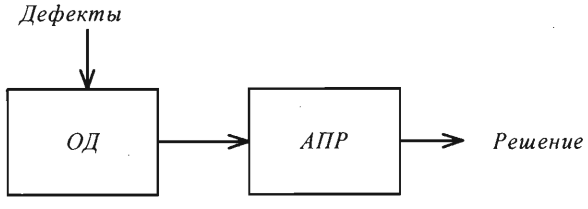
$$G(n)=\max\left\{0,G(n-1)+\operatorname{sign}\left(z(n)-\mu_1+\upsilon/2\right)\right\},\ \mu_1>\mu_2;$$

$$G(0)=0;$$

$$G(n)<h \Rightarrow \text{дефекта нет};$$

$$G(n)\geq h \Rightarrow \text{дефект},$$

где μ_1 и μ_2 – математическое ожидание процесса до и после появления дефекта; h – порог срабатывания алгоритма; υ – порог чувствительности алгоритма.



■ ***Рис. 1.*** *Общая структура системы диагностирования:*
ОД – объект диагностирования;
АПР – алгоритм принятия решений

Алгоритм Гиршика–Рубина–Ширяева:

$$G(n) = e^{z_n}\left(1 + G(n-1)\right);$$

$$G(0) = 0;$$

$$G(n) < h \Rightarrow \text{дефекта нет};$$

$$G(n) \geq h \Rightarrow \text{дефект}.$$

Описание объекта диагностирования

Автоматизированная система контроля уровня воды ОВУ-214 («Водоотливная станция») состоит из трех подсистем: «Сборная емкость», «Исполнительные устройства» и «Энергоснабжение». Подсистема «Сборная емкость» представляет собой резервуар, предназначенный для накопления грунтовых и технических вод, попадающих в тоннель метро. При достижении определенного уровня вода перекачивается из резервуара в канализационную систему, находящуюся вблизи поверхности земли. Резервуар имеет площадь 100 кв. м и высоту 3 м. Скорость прибытия воды непостоянна, но в штатных условиях не превышает 0,5 м/ч. Процесс изменения уровня воды можно представить в виде чередования двух процессов: нарастания уровня воды, когда все насосы отключены, и убывания воды при включении насосов. Оба эти процесса являются инерционными. Так, при дискретности изменения уровня, равной 1 мм, один и тот же уровень сохраняется в течение 1 с или 10 измерений. Данное описание позволяет создать модель процесса изменения уровня воды.

Уровень воды в сборной емкости измеряется с помощью трех датчиков – двух аналоговых и одного дискретного. В качестве модели диагностирования возьмем модель аналогового датчика, в качестве пространства диагностических признаков будем рассматривать разницу показаний двух аналоговых датчиков.

Модель аналогового датчика уровня воды

В качестве модели аналогового датчика предлагается модель нормального распределения погрешности его показаний относительно реального уровня воды:

$$x = h + \varepsilon_{ад},$$

где x – показания аналогового датчика; h – текущий уровень воды; $\varepsilon_{ад} \in N\left(s_{ад}, \sigma_{ад}\right)$ – погрешность измерения аналогового датчика (случайная величина с нормальным законом распределения, имеющим параметры: математическое ожидание, равное $s_{ад}$ и дисперсию $\sigma^2_{ад}$); $s_{ад}$ – систематическая погрешность; $\sigma_{ад}$ – случайная составляющая погрешности.

При отсутствии дефекта аналогового датчика систематическая составляющая погрешности $s_{ад} = 0$, а случайная составляющая удовлетворяет следующему соотношению:

$$3\sigma_{ад.ном} \leq 2 \text{ см},$$

или

$$\sigma_{ад.ном} \leq \frac{2 \text{ см}}{3} = 0{,}67 \text{ см}.$$

Дефектами аналогового датчика являются:

1) появление систематической составляющей погрешности $s_{ад} \neq 0$;

2) увеличение уровня дисперсии $\sigma^2_{ад}$ погрешности датчика.

Применение алгоритмов диагностирования

Будем рассматривать три вида дефектов. Дефекты 1 и 2 – увеличение математического ожидания погрешности измерения одного из аналоговых датчиков на 1 и 3 см соответственно. Дефект 3 – увеличение уровня дисперсии погрешности измерения одного из аналоговых датчиков вдвое.

Моделирование производится в два этапа.

1. Настройка параметров алгоритма на вероятность ложного обнаружения. Исходный процесс (без дефекта) подается в систему; на выходе ведется подсчет ложных обнаружений. За счет варьирования параметров алгоритма происходит его настройка на заданный уровень вероятности ложного обнаружения.

2. Введение дефекта и определение среднего времени обнаружения. Изменяются параметры (математическое ожидание или дисперсия) шумов на входе в систему; на выходе выявляется смещение момента обнаружения относительно истинного момента дефекта. Это смещение и есть время обнаружения.

Для настройки алгоритмов зададим уровень вероятности ложного обнаружения $P_{ло} = 0{,}01$.

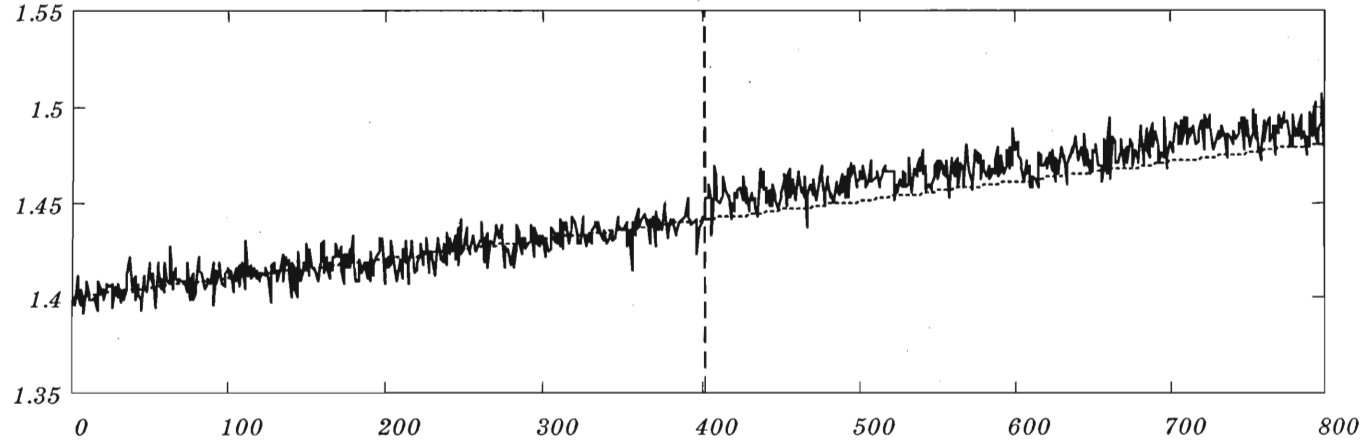
Тогда параметры модифицированного алгоритма кумулятивных сумм: $\upsilon = 0{,}04$; $h = 0{,}9$.

Параметры алгоритма Гиршика–Рубина–Ширяева: $h = 0{,}29$.

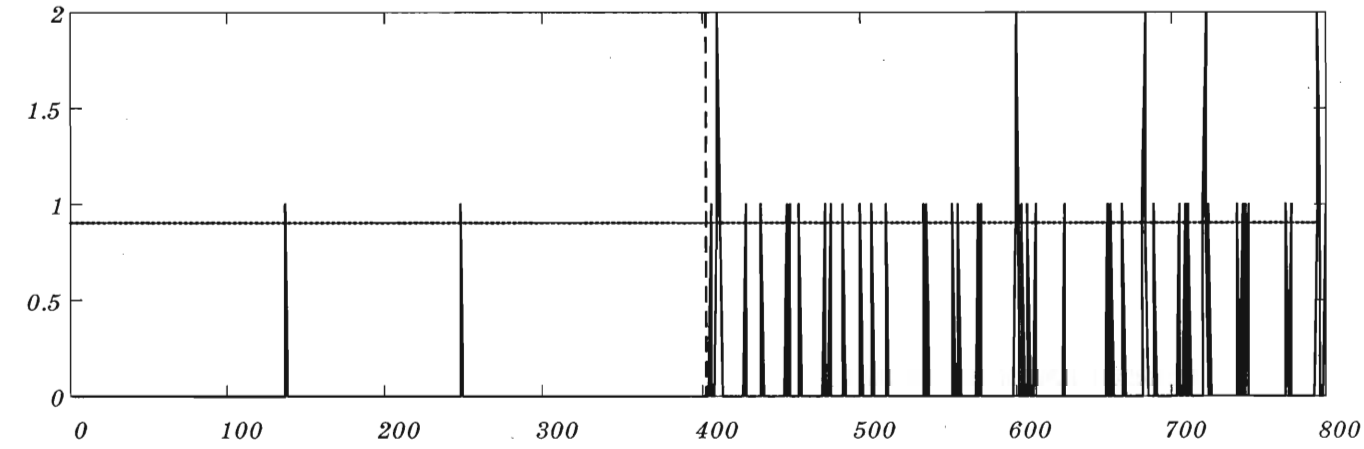
Для моделирования процесса изменения уровня воды, показаний аналоговых датчиков и для реализации алгоритмов диагностирования была написана программа в пакете Matlab (около 600 строк).

Результаты моделирования процесса изменения уровня воды с дефектом 1-го вида (изменение математического ожидания погрешности измерения одного из аналоговых датчиков на 1 см) и соответствующая решающая статистика модифицированного алгоритма кумулятивных сумм показаны на рис. 2 и 3. Время обнаружения $T = 4$.

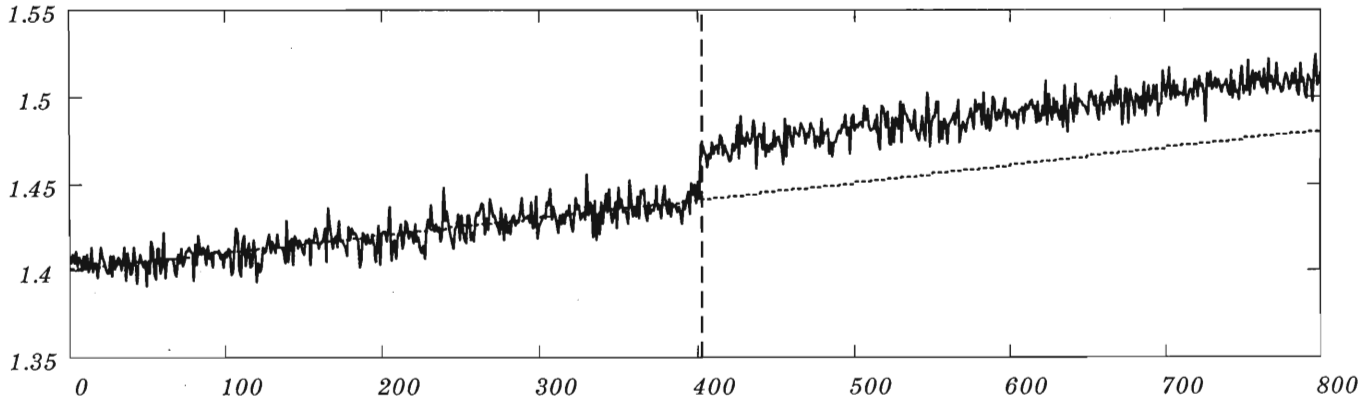
Результаты моделирования процесса изменения уровня воды с дефектом 2-го вида (изменение математического ожидания погрешности измерения одного из аналоговых датчиков на 3 см) и со-



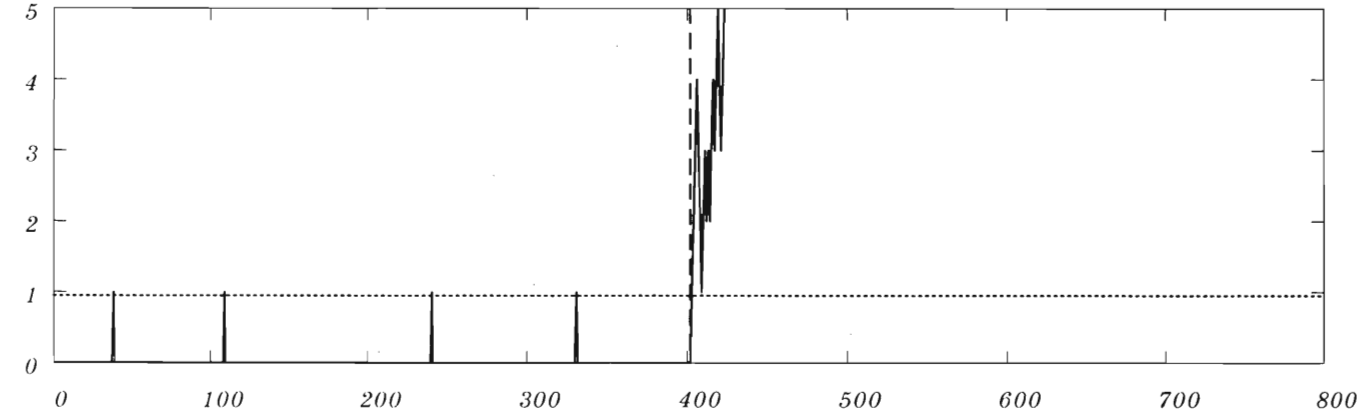
■ *Рис. 2. Моделирование процесса изменения уровня воды с дефектом 1-го вида:*
– – – – – процесс увеличения уровня воды;
——— показания аналогового датчика;
..... момент возникновения дефекта



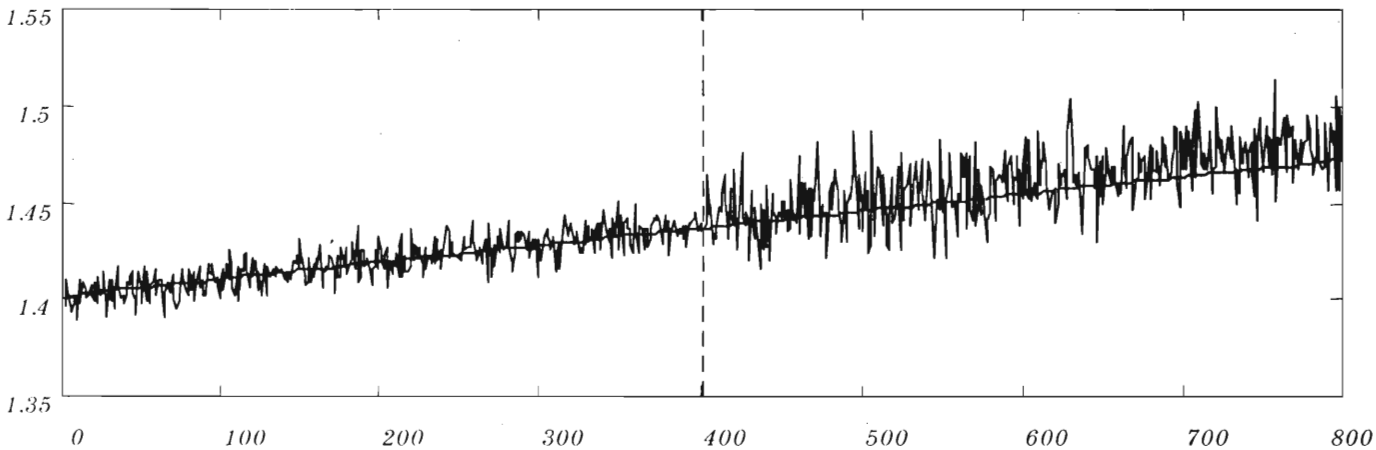
■ *Рис. 3. Решающая статистика алгоритма АКС-м (дефект 1-го вида):*
– – – – – уровень порога срабатывания алгоритма;
——— решающая статистика;
..... момент возникновения дефекта



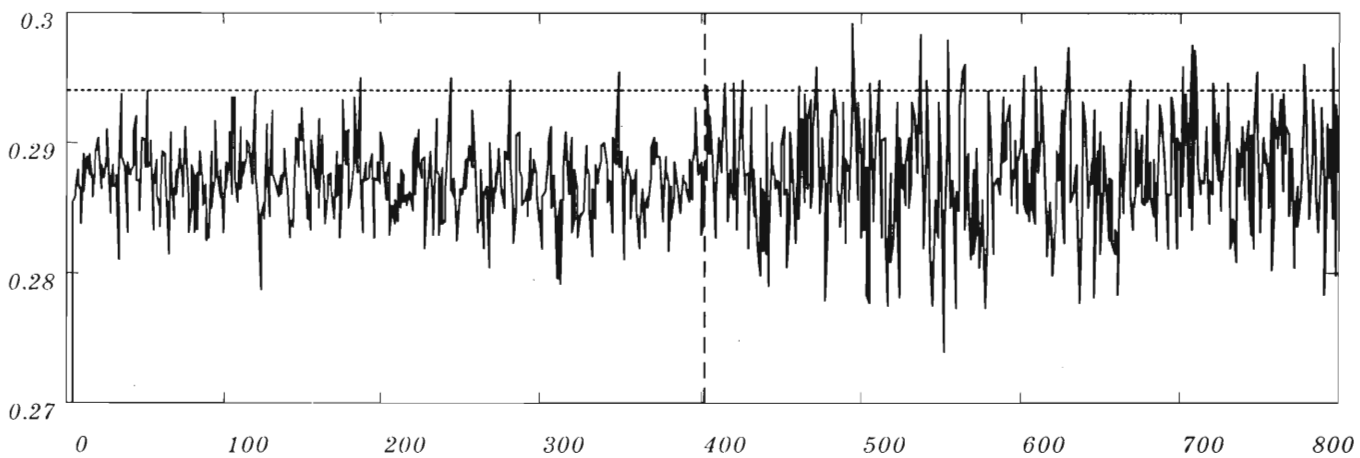
■ *Рис. 4. Моделирование процесса изменения уровня воды с дефектом 2-го вида:*
– – – – – процесс увеличения уровня воды;
——— показания аналогового датчика;
..... момент возникновения дефекта



■ ***Рис. 5.*** *Решающая статистика алгоритма АКС-м (дефект 2-го вида):*
– – – – – уровень порога срабатывания алгоритма;
——— решающая статистика;
..... момент возникновения дефекта



■ ***Рис. 6.*** *Моделирование процесса изменения уровня воды с дефектом 3-го вида:*
– – – – – процесс увеличения уровня воды;
——— показания аналогового датчика;
..... момент возникновения дефекта



■ ***Рис. 7.*** *Решающая статистика алгоритма ГРШ (дефект 3-го вида):*
– – – – – уровень порога срабатывания алгоритма;
——— решающая статистика;
..... момент возникновения дефекта

ответствующая решающая статистика модифицированного алгоритма кумулятивных сумм показаны на рис. 4 и 5. Время обнаружения $T = 2$.

Результаты моделирования процесса изменения уровня воды с дефектом 3-го вида (увеличение уровня дисперсии погрешности измерения одного из аналоговых датчиков вдвое) и соответствующая решающая статистика алгоритма Гиршика–Рубина–Ширяева показаны на рис. 6 и 7. Время обнаружения $T = 3$.

Результаты проведенных исследований позволяют сделать вывод о возможности применения выбранных алгоритмов для обнаружения изменений математического ожидания и дисперсии погрешности измерения аналогового датчика водоотливной станции. При этом величина среднего времени обнаружения дефекта принимает небольшие значения, что подтверждает правильность выбора алгоритмов принятия решений с точки зрения рассматриваемого критерия качества обнаружения (минимум среднего времени обнаружения дефекта при контроле уровня вероятности ложного обнаружения).

Литература

1. **Сабонис С. С.** Обнаружение дефектов в системах управления с использованием фильтра Калмана // Научно-технические ведомости. – 2004. – №1. – С. 214–220.
2. **Бывайков М. Е., Ромащев А. А.** О робастности в задаче обнаружения изменения параметра сдвига случайного процесса // Автоматика и телемеханика. – 1989. – № 7. – С. 138–143.
3. **Бродский Б. Е., Дарховский Б. С.** Проблемы и методы вероятностной диагностики // Автоматика и телемеханика. – 1999. – № 8. – С. 3–50.
4. **Бродский Б. Е., Дарховский Б. С.** Сравнительный анализ некоторых непараметрических методов скорейшего обнаружения момента «разладки» случайной последовательности // Теория вероятностей и ее применения. – 1990. – Т. 35. – № 4. – С. 881–888.
5. **Бендерская Е. Н., Колесников Д. Н., Пахомова В. И.** Функциональная диагностика систем управления: Учеб. пособ. / СПбГТУ. – СПб.: Изд-во СПбГТУ, 2000. – 143 с.
6. **Никифоров И. В.** Модификация и исследование процедуры кумулятивных сумм // Автоматика и телемеханика. – 1980. – № 9. – С. 74–80.
7. **Никифоров И. В.** Последовательное обнаружение изменения свойств стохастических сигналов и систем на основе модифицированного алгоритма кумулятивных сумм // В кн.: Обнаружение изменения свойств сигналов и динамических систем / Под ред. М. Бассвиль, А. Банвениста. – М.: Мир, 1989. – С. 159–187.

УДК 681.518

МЕТОД АВТОМАТИЧЕСКОГО ОТОЖДЕСТВЛЕНИЯ ЛИНИЙ РАВНЫХ ВЫСОТ ПРИ СОЗДАНИИ ЦИФРОВЫХ КАРТ МЕСТНОСТИ НА ОСНОВЕ КАРТОМЕТРИЧЕСКОГО ПОДХОДА

И. А. Зикратов,
канд. техн. наук, доцент
Санкт-Петербургское высшее военное училище радиоэлектроники

Предложен метод, основанный на использовании критерия максимального правдоподобия, позволяющий осуществлять отождествление линий равных высот в процессе оцифровки картографического материала при наличии шумов сканирования. Рассмотрены критерии нахождения узловых точек для аппроксимации линий равных высот ортогональными полиномами и построения точек экстраполяции.

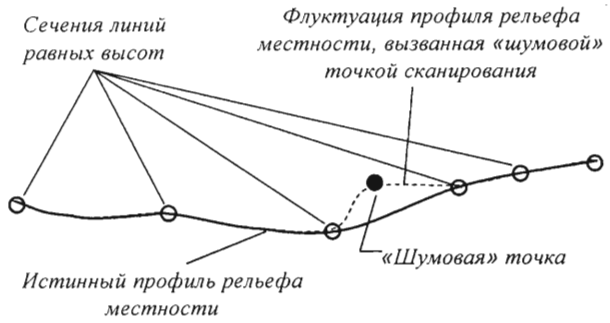
The method based on use of maximal plausibility criterion is offered, allowing carrying out identification of equal heights lines during digitization of cartographical material with scanning noises, is offered. Criteria of central points finding for approximation of equal heights lines by orthogonal polynoms and creation of extrapolation points are considered.

В настоящее время во многих отраслях деятельности человека нашли широкое применение географические информационные системы (ГИС), представляющие собой эффективное средство для анализа задач, в которых в качестве исходных данных используются данные о пространственном положении объектов земной поверхности (географические данные).

Для использования в ГИС географические данные преобразуются в цифровой формат, составляющий основу цифровых карт местности (ЦКМ). В настоящее время наиболее распространенным способом создания ЦКМ является так называемый картометрический подход, суть которого заключается в сканировании бумажного картографического материала и формировании цифровых данных, по содержанию и точности соответствующих исходным топографическим картам. Процесс преобразования данных с бумажных карт в компьютерные файлы называется оцифровкой. Достоинство такого подхода заключается в относительной простоте реализации, однако ему свойственны и некоторые недостатки, затрудняющие практическое использование ЦКМ.

Во-первых, при сканировании бумажной карты неизбежно проявление в той или иной степени неточностей съема информации, заключающихся в образовании случайных разрывов сканируемых линий, или, наоборот, вводе лишних точек, т. е. появлении «шумов» сканирования. Это может приводить к ошибкам при построении профилей рельефа местности, когда сглаживаются существующие или отображаются несуществующие неровности (рис. 1).

Во-вторых, при устранении подобных «шумов» и технологических разрывов линий остается вы-



■ ***Рис. 1.*** *Иллюстрация влияния «шумов» сканирования на формирование профиля рельефа местности*

сокой доля ручного труда при оцифровке данных, что, с одной стороны, является предпосылкой появления ошибок оператора и, с другой стороны, обусловливает повышение себестоимости ЦКМ. Очевидно, что интенсивность «шумов» сканирования можно снизить путем использования качественной бумажной основы, сканеров с высокими характеристиками или совершенствованием программного обеспечения процесса оцифровки, реализующего алгоритмы фильтрации «шумов» сканирования.

Использование подобных алгоритмов позволит осуществить автоматическое распознавание по заданным критериям объектов оцифровки и создание цифровой базы географических данных, сгруппированных по тематическому признаку. При этом снижается степень ручного труда и уменьшается вероятность появления ошибок оператора.

Решение этой задачи может быть реализовано путем анализа графического файла, созданного при сканировании исходной топографической карты. Анализ заключается в последовательном переборе координат пикселей графического файла с определением стандартными библиотечными функциями цвета пикселя, присущего тому или иному объекту (линии равных высот – коричневый, дороги – черный и т. д.), и его взаимного положения по отношению к соседним точкам этого объекта. Такой функцией может, например, служить функция **getPixel()** библиотеки GD языка **Perl**. Результатом координатного и цветового анализа пикселей будет принятие решения о принадлежность координатных точек к тому или иному объекту местности и заполнение соответствующих полей тематической базы данных (ТБД). Очевидно, что при этом должен осуществляться пересчет координатных точек в географическую систему координат.

Совокупность пикселей может образовывать как некие детерминированные фигуры (цифры, топографические знаки, символы, прямоугольники, окружности и т. д.), входящие в состав условных обозначений карты, так и нерегулярные объекты со случайной формой и координатами (линии равных высот, дороги, границы лесных массивов и т. д.).

В первом случае при анализе координатной информации массива пикселей целесообразно использовать методы распознавания образов, которые заключаются в отнесении имеющихся данных к определенному классу с помощью выделения существенных признаков, характеризующих эти данные, из общей массы несущественных деталей [1]. Наличие «шумов» сканирования дает основу для использования также статистических методов распознавания, которые представлены в работах ряда авторов, например [2, 3].

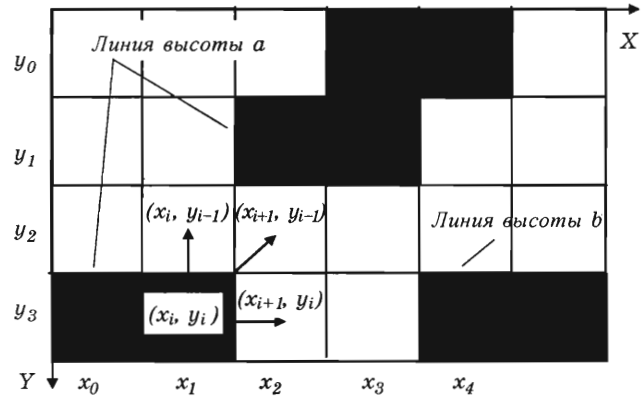
Рассмотрим более подробно формирование тематической базы данных для второго случая, когда необходимо устранить «шумы» сканирования и технологические разрывы при обработке графических данных об объектах с произвольной формой, примером которых могут служить линии равных высот (ЛРВ).

В этом случае, учитывая, что ЛРВ на картах не пересекаются, запись в ТБД информации о текущей координате линии высоты a может осуществляться тогда, когда при изменении текущей координаты графического файла на один пиксель стандартная функция возвращает цвет предшествующего пикселя, что свидетельствует о продолжении ЛРВ в данном направлении. Если же соседние пиксели имеют другой цвет, то записи по данной ЛРВ прерываются. Ближайший найденный пиксель в этом случае может относиться как к линии высоты a, так и к смежной ЛРВ (рис. 2).

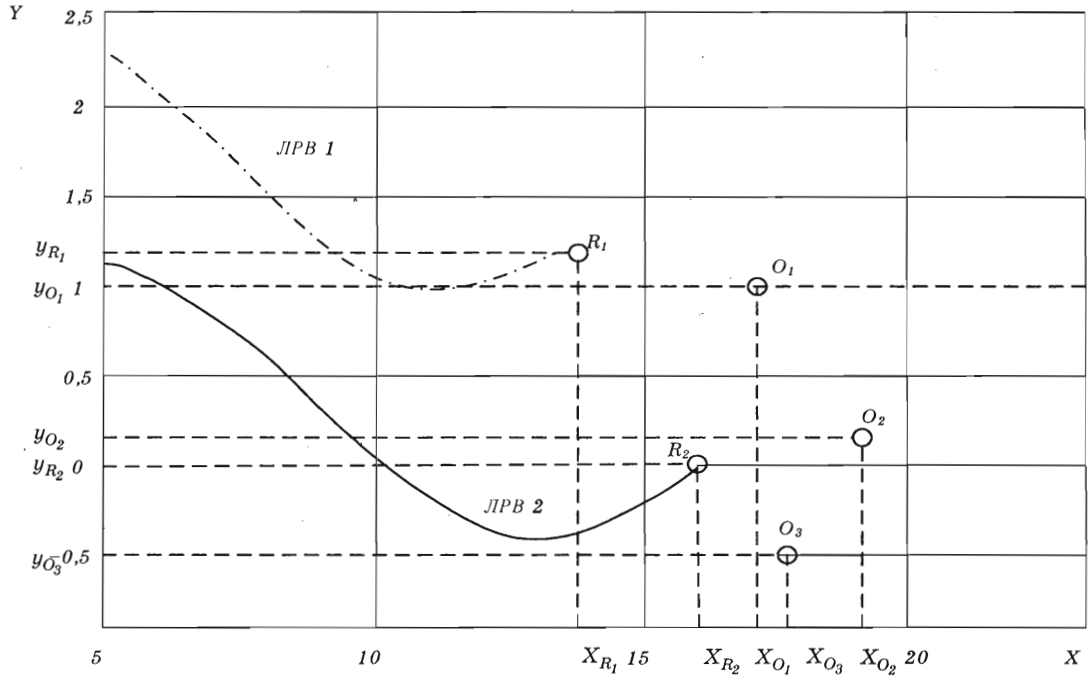
Таким образом, необходимо разработать алгоритм, позволяющий при последовательном анализе пикселей графического файла, относящихся к заданной ЛРВ, в случае обрыва этой линии принять решение, является ли найденная точка продолжением этого объекта. При этом обрыв линии может являться следствием «шума» (см. рис. 2) или технологическим разрывом, служащим для вставки условного знака карты (высоты ЛРВ и др.).

Так как координаты разрывов и «шумовых» точек являются случайными, то разрабатываемый алгоритм должен быть основан на решении статистической задачи отождествления координатных точек. Так, на рис. 3 изображены две ЛРВ, имеющие технологические разрывы, которые служат для вставки значений высот, и «шумовая» точка. Точки разрыва обозначены по принадлежности к ЛРВ буквами R_1 и R_2, отождествляемые точки – O_1, O_2, O_3. При последовательном анализе координат ЛРВ 1–3 для этого случая возможны следующие варианты соотнесения отождествляемых точек к точкам разрыва ЛРВ:

$$R_1 - O_1, \quad R_1 - O_2, \quad R_1 - O_3,$$
$$R_2 - O_1, \quad R_2 - O_2, \quad R_2 - O_3.$$



■ *Рис. 2. Пиксель (x_1, y_3) – точка разрыва ЛРВ, пиксели (x_2, y_1) и (x_4, y_3) – отождествляемые точки*



■ ***Рис. 3.*** *К постановке задачи отождествления*

Логический анализ вариантов отождествления позволяет выявить и исключить из рассмотрения ложные пары отождествляемых точек, при которых происходит пересечение ЛРВ ($R_1 - O_3$ и $R_2 - O_1$), однако оставшиеся варианты отождествления требуют дополнительного анализа:

$$R_1 - O_1 \text{ и } R_2 - O_2 \text{ (1 вариант)},$$
$$R_1 - O_2 \text{ и } R_2 - O_3 \text{ (2 вариант)}.$$

Для решения этой задачи построим точки экстраполяции (ЭТ) функции $f_{ЛРВ1}(x)$, аппроксимирующей функцию ЛРВ 1, и $f_{ЛРВ2}(x)$, аппроксимирующей функцию ЛРВ 2. В рассматриваемой ситуации такими точками будут:

$ЭТ_{11}$ – точка экстраполяции функции $f_{ЛРВ1}\left(x_{O_1}\right)$;

$ЭТ_{12}$ – точка экстраполяции функции $f_{ЛРВ1}\left(x_{O_2}\right)$;

$ЭТ_{22}$ – точка экстраполяции функции $f_{ЛРВ2}\left(x_{O_2}\right)$;

$ЭТ_{23}$ – точка экстраполяции функции $f_{ЛРВ2}\left(x_{O_3}\right)$.

Тогда можно рассмотреть две альтернативные гипотезы отождествления:

гипотеза H_1^* – R_1, O_1 и $ЭТ_{11}$ относятся к ЛРВ 1, а R_2, O_2 и $ЭТ_{22}$ относятся к ЛРВ 2;

гипотеза H_2^* – R_1, O_2 и $ЭТ_{12}$ относятся к ЛРВ 1, а R_2, O_3 и $ЭТ_{23}$ относятся к ЛРВ 2.

Результатом статистического отождествления ЛРВ является выбор одной из альтернативных гипотез. При этом возможны ошибочные решения, характеризующиеся некоторой стоимостью. В данной задаче нет оснований считать, что ошибочные решения имеют различные стоимости, поэтому в качестве критерия оптимального выбора гипотезы может быть использован критерий максимального правдоподобия.

Функции правдоподобия, которые представляют собой совместную плотность вероятности координат точек разрыва и отождествления при условии их принадлежности к соответствующим ЛРВ, с учетом взаимонезависимости этих координат, будут иметь вид:

$$L_{H1} = W\left(x_{R_1}, y_{R_1} / x_{ЭТ_{11}}, y_{ЭТ_{11}}\right) \times$$
$$\times W\left(x_{O_1}, y_{O_1} / x_{ЭТ_{11}}, y_{ЭТ_{11}}\right) \times$$
$$\times W\left(x_{R_2}, y_{R_2} / x_{ЭТ_{22}}, y_{ЭТ_{22}}\right) \times$$
$$\times W\left(x_{O_2}, y_{O_1} / x_{ЭТ_{22}}, y_{ЭТ_{22}}\right); \qquad (1)$$

$$L_{H2} = W\left(x_{R_1}, y_{R_1} / x_{ЭТ_{12}}, y_{ЭТ_{12}}\right) \times$$
$$\times W\left(x_{O_2}, y_{O_2} / x_{ЭТ_{12}}, y_{ЭТ_{12}}\right) \times$$
$$\times W\left(x_{R_2}, y_{R_2} / x_{ЭТ_{23}}, y_{ЭТ_{23}}\right) \times$$
$$\times W\left(x_{O_3}, y_{O_3} / x_{ЭТ_{23}}, y_{ЭТ_{23}}\right). \qquad (2)$$

В выражениях (1), (2) условия принадлежности отождествляемых точек к ЛРВ представляют отклонения их координат от соответствующих ЭТ. Действительно, вероятность принадлежности некоторой отождествляемой точки к ЭТ тем выше, чем меньше их взаимное удаление. Поэтому от условных плотностей вероятности можно перейти к

безусловным, зависящим от взаимного удаления координатных и экстраполяционных точек:

$$L_{\text{Н1}} = W\left(\Delta\mathbf{R}_{R_1/11}\right)W\left(\Delta\mathbf{R}_{O_1/11}\right)\times \\ \times W\left(\Delta\mathbf{R}_{R_2/22}\right)W\left(\Delta\mathbf{R}_{O_2/22}\right); \quad (3)$$

$$L_{\text{Н2}} = W\left(\Delta\mathbf{R}_{R_1/12}\right)W\left(\Delta\mathbf{R}_{O_2/12}\right)\times \\ \times W\left(\Delta\mathbf{R}_{R_2/23}\right)W\left(\Delta\mathbf{R}_{O_3/23}\right), \quad (4)$$

где

$$\Delta\mathbf{R}_{R_i/ij} = \begin{Vmatrix} x_{R_i} - x_{\text{ЭТ}_{ij}} \\ y_{R_i} - y_{\text{ЭТ}_{ij}} \end{Vmatrix} = \begin{Vmatrix} \Delta X_{R_j/ij} \\ \Delta Y_{R_j/ij} \end{Vmatrix}; \quad (5)$$

$$\Delta\mathbf{R}_{O_i/ij} = \begin{Vmatrix} x_{O_i} - x_{\text{ЭТ}_{ij}} \\ y_{O_i} - y_{\text{ЭТ}_{ij}} \end{Vmatrix} = \begin{Vmatrix} \Delta X_{O_j/ij} \\ \Delta Y_{O_j/ij} \end{Vmatrix}. \quad (6)$$

Из выражений (3)–(6) следует, что условия принадлежности координатных точек R_i и O_i к ЛРВ представляют отклонения их координат от соответствующих ЭТ. Действительно, вероятность принадлежности координатной точки к ЭТ тем выше, чем меньше их взаимное удаление.

Отсюда можно сделать вывод, что алгоритм построения точек экстраполяции существенно определяет степень достоверности решения задачи отождествления. Для разработки такого алгоритма рассмотрим ЛРВ как некую функцию $y = f(x)$, заданную таблично на дискретном множестве точек $x_0, x_1, \ldots, x_m$. В данном случае относительно ЛРВ известны только дискретные значения этой функции, и чтобы вычислить другие ее значения за неким отрезком (a, b) узловых точек (точки экстраполяции), ее необходимо приблизить аппроксимирующей функцией $f_{\text{ЛРВ1,2}}(x)$.

Учитывая, что при наличии случайных ошибок в значениях функций, какие имеются при сканировании ЛРВ, предпочтительно применять «сглаживающую» аппроксимацию, которая малочувствительна к шумам, целесообразно воспользоваться представлением аппроксимирующих функций ортогональными многочленами вида [4]:

$$f_{\text{ЛРВ}}(x) = a_0\varphi_0(x) + a_1\varphi_1(x) + \ldots + a_n\varphi_n(x), \quad (7)$$

где $\varphi_k(x)$ – многочлены степени k, попарно ортогональные на заданном множестве точек, получаемые методом Грама – Шмидта, а коэффициенты a_k определяются по формулам

$$a_k = \frac{\sum_{j=0}^{m}\gamma_j f(x_j)\varphi_k(x_j)}{\sum_{j=0}^{m}\gamma_j\varphi_k^2(x_j)}, \quad k = 0, 1, 2, \ldots, n; \quad n \le m.$$

В этом выражении γ_j представляют собой заданные положительные веса, которые в рассматриваемой задаче можно принять равными единице.

Отрезок (a,b), на котором задаются n узловых точек, выбирается из следующих соображений. Линии равных высот представляют собой кривые со случайными отклонениями от некоторого среднего уровня. Корреляцию точек ЛРВ можно качественно описывать с помощью радиуса корреляции – характерного расстояния, на котором корреляционная функция существенно меняется. Так, для гауссовой корреляции таким расстоянием может быть разнесение координат, при котором корреляционная функция уменьшается в e раз. В работе [5] приведены и другие критерии для вычисления величины l случайного поля. В общем случае эта величина не является однозначно определенной – одной и той же корреляционной функции можно сопоставить несколько радиусов корреляции. Однако для одного и того же участка местности часть флуктуаций ЛРВ может отображаться либо не отображаться на карте в зависимости от картографического масштаба, что сужает возможный разброс величины l при дальнейшем анализе оцифрованного материала. В этом случае размеры отрезка (a, b) можно задавать исходя из величины радиуса корреляции ЛРВ.

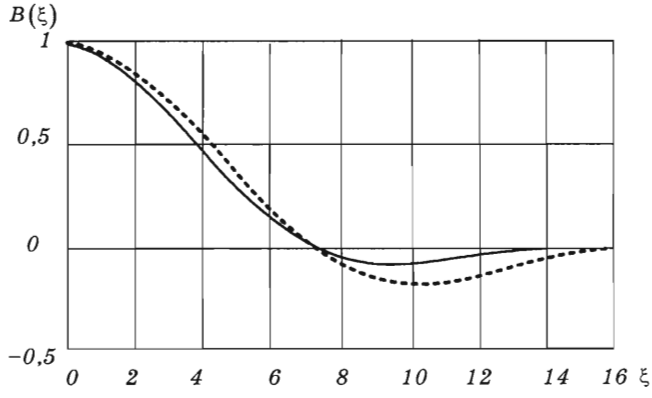
Корреляционные функции ЛРВ 1 и ЛРВ 2 при условии стационарности $f_{1,2}(x)$ имеют вид

$$B_1(\xi_1) = \overline{f_1(x)f_1(x+\xi_1)};$$

$$B_2(\xi_2) = \overline{f_2(x)f_2(x+\xi_2)}.$$

где ξ_1 и ξ_2 – величины сдвига между координатными точками соответствующих функций. Для рассматриваемого примера графики корреляционных функций представлены на рис. 4.

Из графиков видно, что смежные ЛРВ имеют схожие функции корреляции. Если радиус корреляции определять по уровню 0,5, то $l_1 = 3{,}72$, а $l_2 = 4{,}09$.



■ ***Рис. 4.*** *Нормированные корреляционные функции ЛРВ 1 и ЛРВ 2:*
——— *КФ ЛРВ 1;*
------- *КФ ЛРВ 2*

При определении $f_{ЛРВ1,2}(x)$ по формуле (7) для снижения объема вычислений удобно задавать равноудаленные узловые точки, лежащие в пределах отрезков (a_1,b_1) для ЛРВ 1 и (a_2,b_2), где $a_i = x_{R_i} - l_i$, $b_i = x_{R_i}$. Так, при пяти узловых точках выражения для ортогональных многочленов $\varphi_k(x)$ имеют сравнительно простой вид [4]:

$$\varphi_k(t) = \begin{cases} 1 \\ \dfrac{1\cdot t}{2} \\ \dfrac{1}{2}\left(t^2-2\right) \\ \dfrac{1}{6}\left(5t^3-17t\right) \\ \dfrac{1}{12}\left(35t^4-155t^2+72\right) \end{cases},$$

где $t = \dfrac{2x-a-b}{b-a}M$, $2M$ – количество равных частей (для пяти узловых точек $M = 2$).

Результаты вычислений точек экстраполяции представлены на рис. 5. Вычисленные по формулам (5) и (6) величины подставляются в выражения для плотностей вероятности (3) и (4) и в соответствии с критерием максимального правдоподобия принимается решение о выборе гипотезы. Если рассматривать $\Delta\mathbf{R}_{R_i/ij}$ и $\Delta\mathbf{R}_{O_i/ij}$ как результат аддитивного воздействия достаточно большого количества независимых случайных факторов, обусловленных погрешностями в изготовлении исходного картографического материала, его оцифровке и пересчете координат, то соответствующие распределения $W\left(\Delta\mathbf{R}_{M_j/ij}\right)$ можно полагать гауссовыми. Корреляционные матрицы данных распределений имеют вид

$$K_{M_j/ij} = \begin{Vmatrix} \sigma^2_{Mx} & K_{Mxy} \\ K_{Mxy} & \sigma^2_{My} \end{Vmatrix},$$

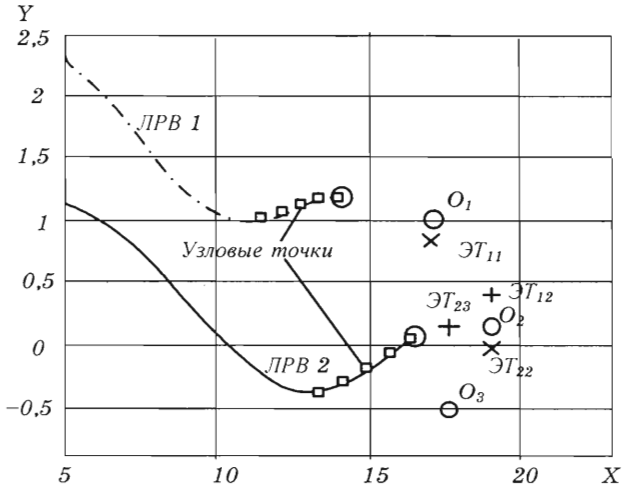
где $M \equiv (R,O)$; σ^2_{Mx} – дисперсия случайной величины $\Delta X_{M_j/ij}$; σ^2_{My} – дисперсия случайной величины $\Delta Y_{M_j/ij}$; K_{Mxy} – корреляционный момент $\Delta X_{M_j/ij}$ и $\Delta Y_{M_j/ij}$ вектора $\Delta\mathbf{R}_{M_j/ij}$.

Тогда $W\left(\Delta\mathbf{R}_{M_j/ij}\right)$ можно представить выражением

$$W\left(\Delta\mathbf{R}_{M_j/ij}\right) = C\exp\left\{-Q_{M_j/ij}\right\}, \qquad (8)$$

где $C = \dfrac{1}{2\pi\left|K_{M_j/ij}\right|^{1/2}}$ – константа; $Q_{M_j/ij} = \dfrac{1}{2\left|K_{M_j/ij}\right|}\times$

$\times\left(\sigma^2_{My}\Delta X^2_{M_j/ij} - 2K_{Mxy}\Delta X_{M_j/ij}\Delta Y_{M_j/ij} + \sigma^2_{Mx}\Delta Y^2_{M_j/ij}\right)$ – квадратичная форма.



■ *Рис. 5. Определение точек экстраполяции*

С учетом выражения (8) можно преобразовать выражения функций правдоподобия (3) и (4) к виду

$$L_{H1} = C\exp\{-Q_{H1}\}; \qquad (9)$$

$$L_{H2} = C\exp\{-Q_{H2}\}, \qquad (10)$$

где Q_{Hj} – сумма квадратичных форм.

Таким образом, из функций (9) и (10) максимальное значение принимает та, которая имеет минимальную сумму квадратичных форм, следовательно, выбор гипотезы отождествления сводится к оценке значений Q_{Hj} и поиску наименьшего из них. Для рассматриваемого примера выбирается гипотеза H_1^*, так как $Q_{H1} < Q_{H2}$.

Решающее правило отождествления можно распространить на произвольное число ЛРВ и отождествляемых точек. Очевидно, что предшествующий логический анализ по выявлению неприемлемых вариантов отождествления, когда происходит пересечение ЛРВ, как и в представленной задаче, позволяет сократить число рассматриваемых гипотез, что приведет к сокращению времени вычисления.

Литература

1. **Ту Дж., Гонсалес Р.** Принципы распознавания образов: Пер. с англ. – М.: Мир, 1978. – 416 с.
2. **Горелик А. Л., Скрипкин В. А.** Методы распознавания. – М.: Высшая школа, 1984. – 208 с.
3. **Фу К.** Последовательные методы в распознавании образов и обучении машин: Пер. с англ. – М.: Наука, 1971. – 255 с.
4. **Корн Г., Корн Т.** Справочник по математике. – М.: Наука, 1974. – 831 с.
5. **Басс Ф. Г., Фукс И. М.** Рассеяние волн на статистически неровной поверхности. – М.: Наука, 1972. – 424 с.

УДК 65.012.122

СОСТАВЛЕНИЕ РАСПИСАНИЙ РЕШЕНИЯ ЗАДАЧ В КОНВЕЙЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Н. В. Колесов,
доктор техн. наук, профессор
М. В. Толмачева,
ведущий инженер
ГНЦ РФ ЦНИИ «Электроприбор» (Санкт-Петербург)

Рассматриваются алгоритмы составления расписаний в конвейерных вычислительных системах. Анализируются три базовых случая, для которых указываются бесперебо́рные алгоритмы составления расписаний для случаев, когда времена решения задач точно известны и когда они задаются интервалами.

The algorithms of Schedule tabling for tasks solution in pipeline computer systems are considered. Three basic cases are analyzed. Author proposes algorithms, which are used for the cases with certain time of task solution specified as intervals.

Введение

Проблема составления расписаний возникает во многих приложениях и, в частности, при формировании последовательности исполняемых задач в многопроцессорных вычислительных системах. Эти системы могут характеризоваться различной структурой – последовательной, параллельной, последовательно-параллельной, параллельно-последовательной. В каждом случае задача имеет свои особенности. Ниже рассматриваются системы с последовательной структурой, или конвейерные системы. Наиболее часто проблема формулируется как поиск такой упорядоченности (расписания) для заданного списка задач, при которой время решения всех задач из списка минимально. Известно [1–3], что решение таких задач в общем случае характеризуется высокой алгоритмической сложностью, описываемой экспоненциальной функцией от длины списка. Такие задачи называют задачами неполиномиальной (экспоненциальной) сложности, или коротко NP-сложными задачами. Однако в некоторых так называемых разрешимых случаях можно предложить более простые алгоритмы. Ряд таких случаев описан, например, в книге [2]. В настоящей статье также анализируются некоторые из таких разрешимых случаев, рассматриваемые при произвольном числе процессоров и задач. Задачи рассматриваются как при условии, что времена решения задач известны точно, так и при условии, что они известны приблизительно и задаются временными интервалами. При этом показывается возможность приближенного сведения задач из расширенного класса к рассматриваемым базовым разрешимым случаям.

Постановка задачи. Базовые разрешимые случаи

Далее будем предполагать, что рассматриваемая вычислительная система представляет собой конвейер из m процессоров $L = \{L_1, L_2, \ldots, L_m\}$, на вход которого поступает последовательность $(J_1, J_2, \ldots, J_n)$ из n задач. Каждая задача разбивается на m фрагментов (по числу процессоров). При этом i-й фрагмент j-й задачи решается на i-м процессоре за время $\tau_{i,j}$ $i = 1, m$, $j = 1, n$. Проблема состоит в определении такой последовательности (расписания) решения задач, которая требует минимального суммарного времени.

Введем на множестве процессоров отношение доминирования «$>$».

Определение 1. Процессор L_q доминирует над процессором L_r ($L_q > L_r$), если $\min\limits_j \tau_{q,j} \geq \max\limits_j \tau_{r,j}$.

Рассмотрим три базовых случая составления расписаний. Во всех случаях на множестве процессоров можно выделить возрастающую или убывающую по отношению доминирования последовательность.

Случай 1. Множество процессоров представляет собой последовательность, убывающую по отношению доминирования, а именно:

$$L_1 > L_2 > \ldots > L_m. \quad (1)$$

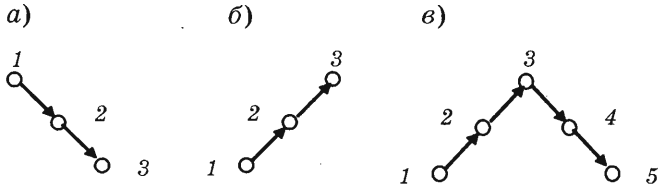
Случай 2. Множество процессоров представляет собой последовательность, возрастающую по отношению доминирования, а именно:

$$L_1 < L_2 < \ldots < L_m. \quad (2)$$

Случай 3. Множество процессоров состоит из пары соединенных последовательностей, первая из которых возрастает, а вторая убывает по отношению доминирования, а именно:

$$L_1 < L_2 < \ldots < L_h > \ldots > L_{m-1} > L_m,\ 1 \le h \le m. \quad (3)$$

На рис. 1 для наглядности приведена графическая интерпретация рассматриваемых случаев. При этом вершины графа отображают процессоры, ребра указывают направленность конвейера, а отношение доминирования между процессорами отражается взаимным расположением соответствующих вершин (доминирующая вершина располагается выше).



■ ***Рис. 1.*** *Графическая интерпретация базовых случаев (а – случай 1, б – случай 2, в – случай 3)*

Решение детерминированной задачи для базовых случаев

Покажем теперь, как сконструировать расписание в первом случае.

Алгоритм 1.

1. Выделим задачу J_s, которая удовлетворяет условию

$$\sum_{i=2}^{m} \tau_{i,s} = \min_j \{\sum_{i=2}^{m} \tau_{i,j}\}.$$

2. Сформируем расписание $\pi_1 = [\tilde{\pi}\, J_s]$, где $\tilde{\pi}$ – произвольное расписание для $(n-1)$ задач, не содержащее задачи J_s.

Покажем оптимальность алгоритма 1. Предварительно отметим важную особенность рассматриваемого случая, заключающуюся в том, что ни перед одним из процессоров конвейера никогда не образуется очередь из задач, ожидающих решения. Этот факт следует из определения отношения доминирования, в соответствии с которым в первом случае самая короткая задача будет решена на некотором процессоре за большее время, чем самая длинная задача на любом последующем процессоре.

Запишем выражение для времени реализации произвольного расписания при выполнении условия (1). Учитывая, что перед процессорами не образуется очередей и все задачи переходят с процессора на процессор без задержек, время $T_1(\pi)$ реализации произвольного расписания π можно представить в виде суммы времени ожидания в исходной очереди для последней n-й задачи и времени t_n ее решения. Причем время ожидания будет определяться суммой времен решения первых фрагментов всех задач, кроме последней. В результате имеем:

$$T_1(\pi) = \sum_{j=1}^{n-1} \tau_{1,j} + t_n = \sum_{j=1}^{n-1} \tau_{1,j} + \sum_{i=1}^{m} \tau_{i,n}.$$

Перенося для удобства анализа первое слагаемое из второй суммы в первую, получаем:

$$T_1(\pi) = \sum_{j=1}^{n} \tau_{1,j} + \sum_{i=2}^{m} \tau_{i,n}. \quad (4)$$

Значение первой суммы представляет собой суммарное время, затрачиваемое на решение первых фрагментов всех задач. Очевидно, что для заданной очереди ее величина фиксирована и не зависит от выбора расписания. В отличие от первой суммы вторая зависит от выбора расписания. Причем оптимальным будет то расписание, которое характеризуется минимальным значением этой суммы. Отсюда следует, что расписание, формируемое в алгоритме 1, является оптимальным для случая 1.

Отметим два существенных обстоятельства. Во-первых, в данном алгоритме отсутствует перебор вариантов расписаний. Во-вторых, для оптимальности расписания достаточно лишь правильного выбора последней исполняемой задачи, которая должна быть наиболее быстро решаемой на всех процессорах, возможно, кроме первого. Упорядоченность же остальных задач не влияет на время исполнения расписания.

Рассмотрим правила формирования расписания во втором случае, который в определенном смысле противоположен первому. Он также характеризуется определяющим свойством, но оно заключается в том, что перед каждым из процессоров всегда образуется очередь из фрагментов задач, ожидающих решения. В результате каждая задача, кроме первой, будет стартовать на любом процессоре с некоторой задержкой относительно момента ее прибытия на вход процессора, поскольку любой i-й фрагмент j-й задачи будет заканчиваться позже, нежели $(i-1)$-й фрагмент следующей за ней $(j+1)$-й задачи. Этот факт также следует из определения отношения доминирования. Алгоритм конструирования расписания в этом случае будет выглядеть следующим образом.

Алгоритм 2.

1. Выделим задачу J_l, которая удовлетворяет условию

$$\sum_{i=1}^{m-1} \tau_{il} = \min_j \{\sum_{i=1}^{m-1} \tau_{i,j}\}.$$

2. Сформируем расписание $\pi_2 = [J_l \tilde{\pi}]$, где $\tilde{\pi}$ – произвольное расписание для $(n-1)$ задач, не содержащее задачи J_l.

Покажем оптимальность этого алгоритма. Запишем выражение для времени реализации произвольного расписания при выполнении условия (2). Учтем, что перед последним n-м процессором (как и перед всеми другими) всегда существует очередь, а значит, этот процессор всегда занят с момента появления на его входе первой и до момента ухода с него последней задачи. В связи с этим время $T_2(\pi)$ реализации произвольного расписания π можно представить в виде суммы времени t_1, необходимого для решения первой задачи, и времени решения последних m-х фрагментов всех остальных задач:

$$T_2(\pi) = t_1 + \sum_{j=2}^{n} \tau_{m,j} = \sum_{i=1}^{m} \tau_{i,1} + \sum_{j=2}^{n} \tau_{m,j}.$$

Перенося для удобства анализа последнее слагаемое из первой суммы во вторую, получаем:

$$T_2(\pi) = \sum_{i=1}^{m-1} \tau_{i,1} + \sum_{j=1}^{n} \tau_{m,j}. \qquad (5)$$

Значение второй суммы представляет собой суммарное время, необходимое для решения последних фрагментов всех задач. Очевидно, что для заданной очереди ее величина фиксирована и не зависит от выбора расписания. В отличие от второй суммы первая зависит от выбора расписания. Причем оптимальным будет то расписание, которое характеризуется минимальным значением этой суммы. Отсюда следует, что расписание, формируемое в алгоритме 2, является оптимальным для случая 2.

Так же, как и в предыдущем случае, в данном алгоритме отсутствует перебор вариантов расписаний, а для оптимальности расписания достаточно лишь правильного выбора первой исполняемой задачи, которая должна быть наиболее быстро решаемой на всех процессорах, возможно, кроме последнего. Упорядоченность же остальных задач не влияет на время исполнения расписания.

Рассмотрим задачу построения оптимального расписания в случае 3.

Алгоритм 3.

1. Выделим задачу J_s, которая удовлетворяет условию

$$\sum_{i=1}^{h-1} \tau_{i,s} = \min_j \{\sum_{i=1}^{h-1} \tau_{i,j}\}.$$

2. Выделим задачу J_p, которая удовлетворяет условию

$$\sum_{i=h+1}^{m} \tau_{i,p} = \min_{\substack{j \\ j \neq s}} \{\sum_{i=h+1}^{m} \tau_{i,j}\}.$$

3. Сформируем расписание $\pi_3 = [J_s \tilde{\pi} J_p]$, где $\tilde{\pi}$ – произвольное расписание для $(n-2)$ задач, не содержащее задач J_s и J_p.

4. Сформируем расписание $\pi_4 = [J_q \tilde{\pi} J_r]$, повторив пп. 1–3, но выполнив пп. 1 и 2 в другой последовательности.

5. Выберем из расписаний π_3 и π_4 наилучшее.

Покажем оптимальность этого алгоритма. Разобьем конвейер на две части. В первую часть включим процессоры с первого по $(h-1)$-й, а во вторую – с h-го по m-й. Очевидно, что первая часть конвейера соответствует случаю 2. В результате каждая задача перед решением на любом процессоре в этой части конвейера попадает в очередь. Поскольку то же самое справедливо и для h-го процессора, можно утверждать, что вторая часть конвейера соответствует случаю 1. Действительно, не только выполняется условие (1), но и задачи выходят на эту часть конвейера без каких-либо дополнительных задержек так, как будто они все одновременно стоят в очереди к h-му процессору. Запишем время $T_3(\pi)$ исполнения произвольного расписания на рассматриваемом конвейере. Представим искомое время $T_3(\pi)$ как сумму двух величин. Первая величина – это время $t_n(1, h-1)$ от начала решения первой задачи на первом процессоре до начала решения n-й задачи на h-м процессоре, вторая величина – это время $t_n(h, m)$ решения n-й задачи на второй части конвейера:

$$T_3(\pi) = t_n(1, h-1) + t_n(h, m).$$

Очевидно, что время $t_n(1, h-1)$ будет равняться времени решения $(n-1)$-й задачи на процессорах с 1-го по h-й, которое определяется выражением (5) и имеет вид

$$t_n(1, h-1) = t_{n-1}(1, h) = \sum_{i=1}^{h} \tau_{i,1} + \sum_{j=2}^{n-1} \tau_{h,j}.$$

Время $t_n(h,m)$ решения n-й задачи на второй части конвейера определяется очевидным выражением

$$t_n(h, m) = \sum_{i=h}^{m} \tau_{i,n}.$$

В результате для $T_3(\pi)$ получаем:

$$T_3(\pi) = \sum_{i=1}^{h} \tau_{i,1} + \sum_{j=2}^{n-1} \tau_{h,j} + \sum_{i=h}^{m} \tau_{i,n}.$$

Перенося для удобства анализа последнее слагаемое из первой суммы и первое слагаемое из третьей суммы во вторую, получаем:

$$T_3(\pi) = \sum_{i=1}^{h-1} \tau_{i,1} + \sum_{j=1}^{n} \tau_{h,j} + \sum_{i=h+1}^{m} \tau_{i,n}. \qquad (6)$$

Значение второй суммы представляет собой суммарное время, затрачиваемое на решение всех задач на h-м процессоре. Очевидно, что для заданной очереди ее величина фиксирована и не зависит от выбора расписания. Первая и третья суммы, в отличие от второй, зависят от выбора расписания. Причем оптимальным будет то расписание, которое характеризуется минимальным значением этой части выражения. Минимальность значения, в свою очередь, будет достигаться, если в расписании первой будет решаться задача, требующая наименьшего времени для первых $(h-1)$ фрагментов, а последней – задача, наиболее быстро решаемая на последних $(m-h)$ процессорах. Поскольку одна и та же задача может удовлетворять обоим требованиям, надо попробовать разместить ее на каждой из этих позиций, а затем выбрать наилучшее из этих двух расписаний. Отсюда следует, что расписание, формируемое в алгоритме 3, является оптимальным для случая 3.

Отметим, что в данном алгоритме перебор расписаний также практически отсутствует. При построении расписания достаточно проанализировать лишь два возможных варианта, а для оптимальности расписания достаточно лишь правильного выбора первой и последней исполняемых задач. Упорядоченность же остальных задач не влияет на время исполнения расписания.

Пример. Построим расписание для конвейерной системы, для которой времена решения задач, выраженные в условных единицах, приведены в табл. 1.

Нетрудно заметить, что данная система удовлетворяет условию (3). Тогда в соответствии с алгоритмом 3 для составления расписания для каждой задачи следует вычислить две суммы:

$$S_1 = \sum_{i=1}^{h-1} \tau_{i,1}; \quad S_2 = \sum_{i=h+1}^{m} \tau_{i,n}.$$

Результаты вычислений приведены в табл. 2.

Из таблицы видно, что первая сумма принимает минимальное значение в первой задаче J_1, а вторая – в четвертой J_4. Следовательно, эти задачи должны занимать соответственно первую и последнюю позиции в оптимальном расписании. Таким образом, в данном случае имеются два варианта оптимального расписания: $\pi_1 = J_1J_2J_3J_4$ и $\pi_2 = J_1J_3J_2J_4$. На рис. 2 приведен график Ганта, соответствующий построенному оптимальному расписанию π_1. На этом графике по оси ординат отложены номера задач, а по оси абсцисс – время, представленное в условных единицах. Для каждой из задач временные интервалы исполнения ее фрагментов на разных процессорах выделены чередующимися цветами. Как видно, время исполнения всего расписания равно 45, что совпадает с расчетным результатом, полученным из выражения (6).

Составление расписаний в случае задач с неопределенными временами решения

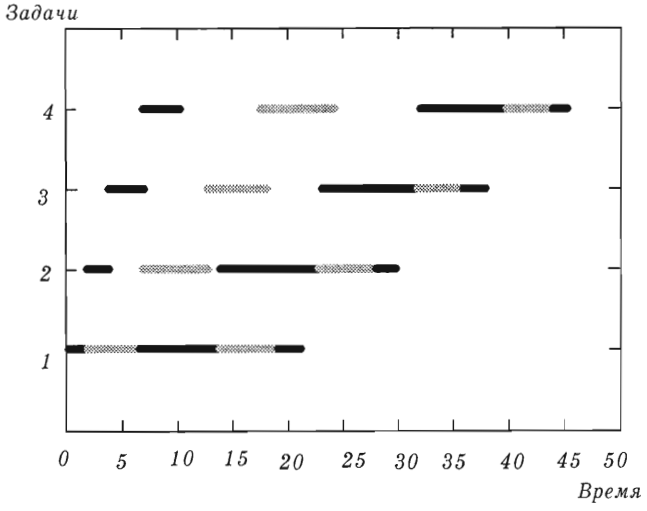
Рассмотрим теперь случай, когда времена решения задач известны приблизительно и задаются временны́ми интервалами $\tilde{\tau}_{ij} = [\underline{\tau}_{ij}, \overline{\tau}_{ij}]$ $i = \overline{1,m}$, $j = \overline{1,n}$, где $\underline{\tau}_{ij}$ и $\overline{\tau}_{ij}$ – нижняя и верхняя границы интервала $\tilde{\tau}_{ij}$ соответственно. Будем называть такую модель многопроцессорной системы недетерминированной системой, а детерминированную

■ *Таблица 1*

Задачи	Процессоры				
	1	2	3	4	5
1	2	5	7	5	2
2	2	6	9	5	1
3	3	5	9	4	2
4	3	6	8	4	1

■ *Таблица 2*

Суммы	Задачи			
	1	2	3	4
S_1	7	8	8	9
S_2	7	6	6	5



■ *Рис. 2. График Ганта для расписания из примера*

модель, получаемую из нее заменой интервальных времен $\tilde{\tau}_{ij} = [\underline{\tau}_{ij}, \overline{\tau}_{ij}]$ $i = \overline{1,m}$, $j = \overline{1,n}$ их нижними $\underline{\tau}_{ij}$ (верхними $\overline{\tau}_{ij}$) границами, – нижней (верхней) системой по отношению к исходной недерминированной системе.

Очевидно, что время $\tilde{T}(\pi)$ исполнения некоторого расписания π интервальной системой будет представлять собой временной интервал, который вычисляется как функция от значений временных интервалов $\tilde{\tau}_{ij} = [\underline{\tau}_{ij}, \overline{\tau}_{ij}]$ $i = \overline{1,m}$, $j = \overline{1,n}$, затрачиваемых на решение задач. При этом используются интервальные арифметические операции [5]:

$$\tilde{\tau}_i + \tilde{\tau}_j = [\underline{\tau}_i, \overline{\tau}_i] + [\underline{\tau}_j, \overline{\tau}_j] = [\underline{\tau}_i + \underline{\tau}_j, \overline{\tau}_i + \overline{\tau}_j];$$

$$\tilde{\tau}_i - \tilde{\tau}_j = [\underline{\tau}_i, \overline{\tau}_i] - [\underline{\tau}_j, \overline{\tau}_j] = [\underline{\tau}_i - \overline{\tau}_j, \overline{\tau}_i - \underline{\tau}_j];$$

$$\tilde{\tau}_i \tilde{\tau}_j = [\underline{\tau}_i, \overline{\tau}_i][\underline{\tau}_j, \overline{\tau}_j] =$$

$$= \left[\min\{\underline{\tau}_i\underline{\tau}_j, \underline{\tau}_i\overline{\tau}_j, \overline{\tau}_i\underline{\tau}_j, \overline{\tau}_i\overline{\tau}_j\},\ \max\{\underline{\tau}_i\underline{\tau}_j, \underline{\tau}_i\overline{\tau}_j, \overline{\tau}_i\underline{\tau}_j, \overline{\tau}_i\overline{\tau}_j\}\right];$$

$$\tilde{\tau}_i / \tilde{\tau}_j = [\underline{\tau}_i, \overline{\tau}_i] / [\underline{\tau}_j, \overline{\tau}_j] = [\underline{\tau}_i, \overline{\tau}_i][1/\overline{\tau}_j, 1/\underline{\tau}_j].$$

При поиске оптимального расписания необходимо сравнивать между собой интервальные значения времен выполнения различных вариантов расписаний. Для этого определим отношение частичного порядка на множестве интервалов.

Определение 2. Интервал $\tilde{\tau}_i = [\underline{\tau}_i, \overline{\tau}_i]$ не меньше интервала $\tilde{\tau}_j = [\underline{\tau}_j, \overline{\tau}_j]$ ($\tilde{\tau}_i \geq \tilde{\tau}_j$), если $\underline{\tau}_i \geq \underline{\tau}_j$ и $\overline{\tau}_i \geq \overline{\tau}_j$.

Известно [4], что время $\tilde{T}(\pi)$ исполнения некоторого расписания π недетерминированной системой равно интервалу $[\underline{T}(\pi), \overline{T}(\pi)]$, где $\underline{T}(\pi)$ и $\overline{T}(\pi)$ – время исполнения этого расписания соответственно для нижней и верхней систем. При этом оптимальное интервальное расписание образуется как пересечение множеств оптимальных расписаний для нижней и верхней систем и не всегда существует. Последнее происходит из-за того, что описания нижней и верхней систем слишком сильно расходятся, в результате чего им соответствуют разные оптимальные расписания. Если поиск оптимального расписания для недетерминированной системы представить как вычисление для каждого расписания времени его исполнения недетерминированной системой с последующим выбором расписания с минимальным временем исполнения, то об отсутствии решения станет известно при сопоставлении интервальных значений времен исполнения. В этом случае окажется, что полученные интервалы не сравнимы между собой. Таким образом, факт отсутствия оптимального решения является платой за недостаток информации.

Введем на множестве процессоров недетерминированной системы отношение доминирования «$\succ$», аналогичное отношению «$>$» для детерминированной системы.

Определение 3. Процессор L_q доминирует над процессором L_r ($L_q \succ L_r$), если $\min\limits_j \underline{\tau}_{q,j} \geq \max\limits_j \overline{\tau}_{r,j}$.

На основе этого отношения можно определить три базовых случая для недетеминированной системы, повторяющие случаи (1)–(3) с точностью до замены отношения «$>$» на отношение «$\succ$».

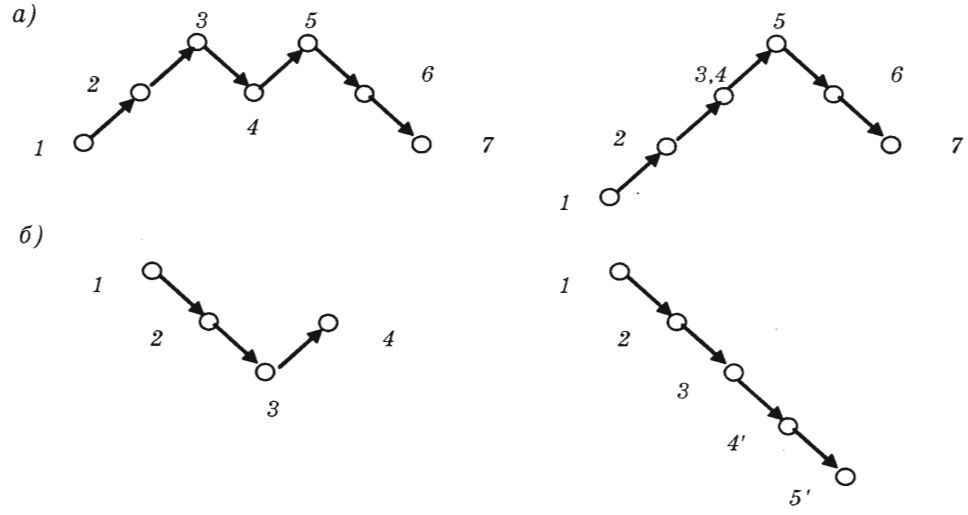
Предположим далее, что во всех рассматриваемых примерах существует оптимальное расписание, т. е. интервальные значения времен исполнения всех расписаний сравнимы между собой. Нетрудно заметить, что все рассуждения, приведенные в предыдущем разделе, а значит, и соотношения (4)–(6), справедливы для любой детерминированной системы, времена исполнения задач в которой принимают значения из соответствующих интервалов недетерминированной системы. На этом основании можно утверждать, что они остаются справедливыми и для недетерминированной системы при условии замены обычных арифметических операций на интервальные.

Приближенные решения с использованием базовых случаев

Как уже отмечалось во введении, в общем случае неизвестны алгоритмы поиска оптимальных расписаний полиномиальной сложности для конвейерной системы. В связи с этим предлагается использовать приближенные решения, предполагающие преобразование задачи к виду, удовлетворяющему одному из условий (1)–(3) для детерминированных систем или аналогичным условиям для недетерминированных систем. Это преобразование не является эквивалентным и в общем случае приводит к решению, отличному от оптимального. Рассмотрим два типа преобразований.

Сущность преобразования первого типа состоит в объединении двух или более последовательных фрагментов всех задач и соответственно в агрегировании исполняющих их процессоров, т. е. в замене двух или более процессоров одним с эквивалентной производительностью. При этом времена работы агрегированных процессоров определяются суммой времен работы объединяемых процессоров. Пример такого преобразования представлен на рис. 3, *а*. В этом примере между собой объединяются фрагменты 3 и 4, причем предполагается, что фрагмент 5 доминирует над объединенным фрагментом. В результате получаем задачу, соответствующую случаю 3.

Сущность преобразования второго типа состоит в расщеплении одного или более последовательных фрагментов всех задач на большее число фрагментов и соответственно в замене одного из процессоров двумя или более виртуальными процессорами. Пример такого преобразования представлен на рис. 3, *б*. В этом примере фрагмент 4 расщепляется на два фрагмента 4′ и 5′. При этом предполагается, что фрагмент 3 доминирует над получаемым в результате расщепления фрагментом 4′, а фраг-



■ *Рис. 3. Примеры преобразований задач*

мент 4′ – над фрагментом 5′. В результате получаем задачу, соответствующую случаю 1. Очевидно, что такое расщепление с точностью до числа образуемых при расщеплении фрагментов всегда возможно.

Каждое из этих преобразований можно охарактеризовать достаточно очевидным, но важным свойством. Для первого преобразования (объединение фрагментов) оно звучит следующим образом.

Свойство 1. Значение времени $\overline{T}_{\text{опт}}(m',n)$ $(m' < m)$ исполнения оптимального расписания, получаемого для преобразованной задачи, является верхней границей для значения времени $T_{\text{опт}}(m,n)$ исполнения оптимального расписания для исходной задачи.

Действительно, в результате первого преобразования параллелизм решения для заданной совокупности задач снижается, а значит, время решения всех задач не может уменьшиться.

Для второго преобразования (расщепление фрагментов) свойство звучит следующим образом.

Свойство 2. Значение времени $\underline{T}_{\text{опт}}(m'',n)$ $(m'' > m)$ исполнения оптимального расписания, получаемого для преобразованной задачи, является нижней границей для значения времени $T_{\text{опт}}(m,n)$ исполнения оптимального расписания для исходной задачи.

Действительно, в результате второго преобразования параллелизм решения для заданной совокупности задач возрастает, а значит, время решения всех задач не может уменьшиться.

Этими свойствами можно руководствоваться при определении правил поиска приближенных решений, а именно, выбирать в качестве решения задачи расписание, оптимальное с точки зрения верхней или нижней границ. При решении задачи может оказаться возможным определение более, чем одной границы. Если в этом случае разные границы предъявляют к расписанию непротиворечивые требования, то они могут быть удовлетворены одновременно.

Заключение

В настоящей статье исследованы вопросы построения расписаний для конвейерных вычислительных систем. При этом проанализированы три базовых случая, для которых указаны по существу беспереборные алгоритмы построения оптимальных расписаний для произвольного числа задач и процессоров. Эти случаи проанализированы как при условии, что времена решения задач точно известны, так и при условии, что эти времена заданы интервалами. Дополнительно рассмотрена возможность получения приближенных решений путем сведения исходной задачи к рассмотренным базовым случаям.

Литература

1. **Конвей Р. В., Максвелл В. Л., Миллер Л. В.** Теория расписаний. – М.: Наука, 1975. – 282 с.
2. **Левин В. И.** Структурно-логические методы исследования сложных систем с применением ЭВМ. – М.: Наука, 1987. – 304 с.
3. **Танаев В. С., Сотсков Ю.Н., Струсевич В.А.** Теория расписаний. Многостадийные системы. – М.: Наука, 1989. – 322 с.
4. **Левин В. И.** Оптимизация расписаний в М-стадийной системе с неопределенными временами обработки // Автоматика и телемеханика. – 2002. – № 2. – С. 129–136.
5. **Калмыков С. А., Шокин Ю. А., Юлдашев З. Х.** Методы интервального анализа. – Новосибирск: Наука, – 1986. – 294 с.

УДК 681.3.06:62-507

РЕШЕНИЕ ЗАДАЧ С ПОМОЩЬЮ КЛЕТОЧНЫХ АВТОМАТОВ ПОСРЕДСТВОМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ CAME&L (Часть I)

Л. А. Наумов,
аспирант
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Работа представляет собой введение в программирование для пакета CAME&L посредством библиотеки CADLib. Описываются основы решения задач с помощью рассматриваемого программного обеспечения, а именно – создание пользовательских правил для клеточных автоматов. На примере игры «Жизнь» демонстрируется разработка простейших решений, использование зональной оптимизации, поддержка многопроцессорной и кластерной вычислительных систем, а также – обобщенных координат. Кроме того, приводится пример решения физической задачи, а именно уравнения теплопроводности.

This work represents the introduction into programming for CAME&L software by means of CADLib library. Bases of building tasks' solutions with the help of considered software, namely the creation of user rules for cellular automata, are described. Here five variants of "Game of Life" are introduced: plain one, variant with zonal optimization, implementations for multiprocessor and cluster computing systems and for generalized coordinates. Moreover the solution of physical problem, thermal conductivity equation, is described.

Введение

Большинство актуальных в настоящее время вычислительных задач требует для своего решения мощностей, превосходящих возможности однопроцессорного персонального компьютера. Это обстоятельство послужило причиной чрезвычайно бурного развития теории, а также программных и аппаратных средств параллельных и распределенных вычислений.

Особой популярностью пользуются реализации стандарта MPI (Message Passing Interface) [1] и библиотека PVM (Parallel Virtual Machine) [2]. Их изучение включено в образовательные программы университетов мира.

Возникла идеология Grid-вычислений [3–5]. Наиболее популярным средством для их организации является пакет Globus Toolkit [6].

Настоящая работа посвящена еще одному средству для параллельных и распределенных вычислений – программному обеспечению CAME&L (Cellular Automata Modeling Environment & Library) [7–9], среде выполнения и библиотеке разработчика клеточных автоматов.

Клеточные автоматы [10] – дискретные динамические системы, поведение которых может быть полностью описано в терминах локальных зависимостей [11]. Спектр их применения чрезвычайно широк [12–14]. Например, они могут быть использованы для моделирования распределенных систем (пространственно-протяженных, распределенных во времени или по другому параметру).

Клеточные автоматы удобны для решения дифференциальных уравнений и уравнений в частных производных, так как эти уравнения описывают непрерывные динамические системы, а, следовательно, рассматриваемые автоматы являются их дискретными аналогами.

Можно считать, что клеточный автомат представляет собой дискретный эквивалент понятия «поле».

Кроме того, рассматриваемые автоматы образуют общую парадигму параллельных вычислений подобно тому, как машины Тьюринга образуют парадигму последовательных вычислений. В результате они могут быть использованы для реализации параллельных алгоритмов, как модели, обладающие естественным параллелизмом.

Было решено разрабатывать пакет CAME&L как средство, предоставляющее развитый инструментарий для решения научных и образовательных задач на основе клеточных автоматов.

При работе на однопроцессорном компьютере это программное обеспечение имеет образовательное значение, как инструмент для выполнения, изучения и визуализации параллельных алгоритмов. Многопроцессорная или кластерная платформы позволят использовать естественный параллелизм клеточных автоматов для осуществления трудоемких вычислений.

Анализ более чем двадцати пяти существующих средств решения задач с помощью клеточных автоматов подтвердил адекватность и необходимость разработки нового программного обеспечения в силу приведенных ниже обстоятельств.

1. Большинство из существующих средств решения задач с помощью клеточных автоматов не удовлетворяют современным требованиям к пользовательскому интерфейсу и не предоставляют возможности, которые стали обычными и неотъемлемыми и на поддержку которых пользователь имеет основания рассчитывать. В настоящее время отсутствуют программные продукты со столь удобным интерфейсом, каким обладает среда, входящая в рассматриваемый пакет. В ней, в частности, реализованы такие возможности, как поддержка буфера обмена; поддержка отмены/повторения последней операции; поддержка печати; возможность сохранения состояния решетки в виде рисунков для иллюстрирования публикаций.

2. Все известные продукты накладывают жесткие ограничения на используемый тип клеточных автоматов. Разрабатываемая среда универсальна и позволяет использовать модели, принадлежащие даже более широкому классу, чем класс «клеточные автоматы».

3. Почти все средства используют различные языки для описания автомата, причем иногда эти языки весьма сложны. Таким образом, появляется еще один навык, которым должен овладеть исследователь прежде, чем приступить к решению задачи.

4. Большинство известных средств моделирования клеточных автоматов разработаны для операционных систем семейства UNIX/Linux. По крайней мере, хорошие экземпляры существуют только для них. Предлагаемое программное обеспечение разработано для операционных систем семейства Windows NT[1]. Однако части, не связанные с пользовательским интерфейсом, не зависят от платформы.

5. Подавляющее большинство существующих продуктов не «моделируют» клеточные автоматы, а только «имитируют» их, так как не используют естественного параллелизма этих моделей, не поддерживают средств для осуществления параллельных или распределенных вычислений. При решении задач на кластерной платформе программное обеспечение CAME&L использует специализированный сетевой протокол CTP (Commands Transfer Protocol) [15, 16], разработанный автором. Поддержка многопроцессорной платформы также предусмотрена.

Проект CAME&L получил высокую оценку экспертов на международной конференции по компьютерным наукам (ICCS-2003), проходившей в Мельбурне и Санкт-Петербурге в 2003 году, а также на шестой международной конференции «Клеточные автоматы в науке и промышленности» (ACRI-2004), проходившей в Амстердаме в 2004 году.

В настоящей работе приводятся примеры решения различных задач с помощью пакета CAME&L.

Основные понятия и свойства пакета CAME&L

Программное обеспечение CAME&L можно разделить на три основные части.

1. Среда выполнения – приложение с богатым и дружественным пользовательским интерфейсом. Базовой абстракцией в ней является понятие «эксперимент», вычислительная задача, описанная в терминах клеточных автоматов. В данном контексте это понятие – синоним термина «документ».

Среда может работать на однопроцессорном компьютере, многопроцессорной системе или в вычислительном кластере (в том числе и через Internet). При этом продукт не требует никаких дополнительных инструментов для организации кластера, так как все необходимые возможности реализованы внутри него.

Среда имеет многодокументный интерфейс для того, чтобы управлять несколькими экспериментами одновременно, а также реализовывать межавтоматные взаимодействия в процессе моделирования.

Помимо этого она обеспечивает сохранение документов в формате XML (файлы с расширением «cml»). Для уменьшения размера этих файлов информация о состоянии решетки автомата может быть сжата внутри документа с помощью алгоритма BZip2.

2. Библиотека разработчика клеточных автоматов CADLib (Cellular Automata Development Library) – богатый набор C++-классов [17], которые служат для создания так называемых «компонентов», элементов решения задач посредством программного обеспечения CAME&L. Библиотека предоставляется для использования и расширения пользователями при разработке своих решений.

Помимо классов, она содержит функции, макроопределения и константы, делающие разработку компонентов настолько простой, насколько это возможно.

Настоящая работа представляет введение в решение задач с помощью библиотеки CADLib, при

[1] Имеются в виду такие операционные системы, как Windows NT 4, Windows 2000, Windows XP и т. д.

этом вопросы, касающиеся работы в среде, рассматриваться не будут.

3. Стандартные компоненты – базовый набор «кирпичиков», из которых могут быть построены решения пользовательских задач.

Каждый клеточный автомат представляется набором компонентов следующих четырех типов.

1. Решетка (grid) – осуществляет визуализацию решетки автомата и навигацию по клеткам.

2. Метрика (metrics) – сопоставляет клеткам координаты, вводит отношение соседства и функции вычисления расстояний между клетками.

3. Хранилище данных (datum) – обеспечивает хранение, обмен и преобразования состояний клеток, а также некоторые аспекты визуализации (соответствие между состоянием клетки и цветами, используемыми для ее отображения).

4. Правила (rules) – описывают итерацию, не только функцию переходов, но и метод разделения задачи на подзадачи, использование оптимизации вычислений, а также многое другое.

Идеология компонентов позволяет «собирать» автоматы с различной функциональностью, посредством незначительных изменений переходить от одной задачи к другой, пробовать решать одну и ту же задачу на разных решетках, с различными метриками и т. п. Кроме того, она предоставляет возможность распределять разработку программного обеспечения эксперимента между различными исполнителями.

Каждый компонент декларирует список собственных параметров, изменение значений которых обеспечивает настройку компонента.

Компонент, реализующий правила автомата, декларирует также список анализируемых параметров, характеризующих работу системы. Среда позволяет наблюдать изменение их значений в динамике, строить графики, файлы отчетов и т. п.

Для анализа таких параметров предусмотрен пятый тип компонентов – анализатор (analyzer).

В стандартный набор компонентов входит специализированный анализатор для построения графиков динамики производительности вычислений, что позволяет настраивать систему для достижения оптимальных результатов.

Реализация метрики в виде отдельного компонента позволяет использовать нестандартные системы координат, как, например, обобщенные координаты [7, 18], четыре вида которых входят в набор стандартных компонентов.

При постановке эксперимента пользователь может воспользоваться компонентами, входящими в базовый набор или разработать свои собственные.

С точки зрения разработчика каждый класс, реализующий компонент – потомок некого класса библиотеки CADLib или класса другого компонента. Иерархия классов компонентов восходит к классу `CAComponent`. От него наследуются классы, соответствующие всем пяти типам компонентов. Базовый класс решеток носит имя `CAGrid`, метрик – `CAMetrics`, хранилищ данных – `CADatum`, правил `CARules`, а анализаторов – `CAAnalyzer`.

Для того чтобы быть использованным в среде, компонент, а также три функции для его аутентификации, создания и удаления помещаются в динамическую библиотеку DLL (Dynamic-Link Library).

Аналогично, иерархия классов, реализующих параметры компонентов, восходит к классу `Parameter`. Анализируемые параметры компонента, реализующего правила автомата, с точки зрения реализации ничем не отличаются от обычных параметров.

Полная диаграмма классов библиотеки CADLib и стандартных компонентов приведена на рис. 1.

Параллелограммы с пунктирными границами обозначают шаблоны классов [17]. Серые фигуры показывают классы стандартных компонентов. Они распространяются вместе с исходным кодом, но формально в библиотеку CADLib не входят. Остальные параллелограммы обозначают обычные классы.

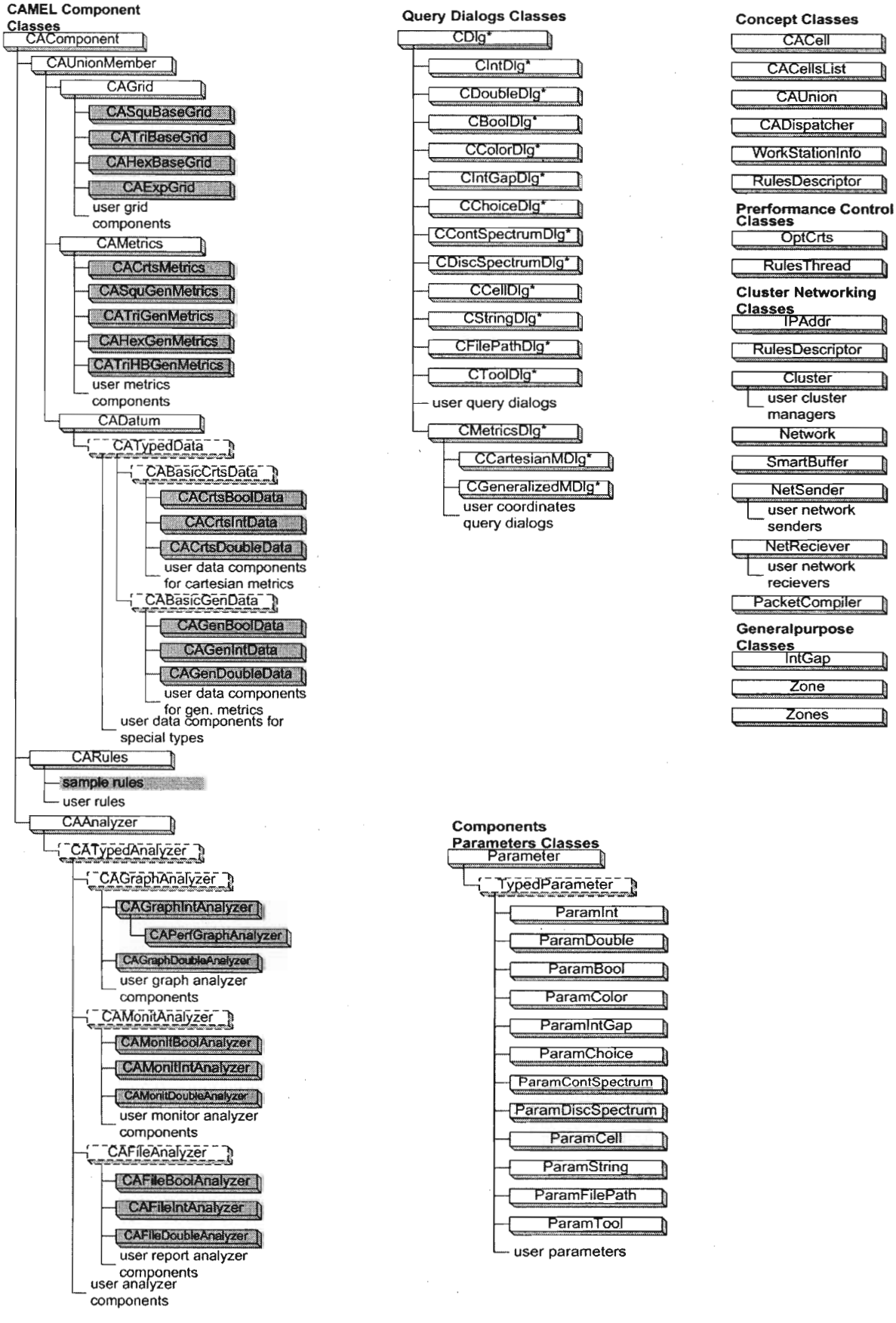
Большинство классов разрабатывалось так, чтобы они не зависели от платформы на уровне исходного кода. В них используются только стандартные библиотеки и библиотека STL (Standard Template Library) [17]. Однако некоторые классы, преимущественно, имеющие отношение к пользовательскому интерфейсу, имеют связи с библиотекой MFC (Microsoft Foundation Classes) [19]. Эти классы отмечены на диаграмме знаком «*». При переносе библиотеки CADLib на другую платформу их необходимо изменить для работы с соответствующей библиотекой.

Кроме того, на диаграмме специально показаны места, которые могут занять пользовательские классы в зависимости от их назначения.

Одним из ключевых типов данных является тип `CACell`, служащий для представления идентификаторов клеток решетки, как 64-разрядных целых чисел.

Необходимо отметить также чрезвычайно востребованный класс `Zone`, описывающий некоторую область на трехмерной решетке автомата. Экземпляр класса хранит шесть значений типа `CACell`, определяющих три отрезка вдоль трех осей пространства `[a1, b1]`, `[a2, b2]` и `[a3, b3]`. В случае необходимости описания зоны на двумерной решетке `a3` полагается равным `b3`. Для одномерного случая, помимо этого, `a2` присваивается `b2`. Кроме того, класс `Zone` реализует большой набор функций зональной алгебры.

Так как в задачи настоящей работы не входит подробный обзор классов библиотеки CADLib, а лишь демонстрация примеров разработки пользовательских компонентов правил клеточных автоматов, рассмотрим лишь необходимые для дальнейшего изложения классы `CAComponent` и `CARules`, а также приведем некоторые начальные сведения о параметрах и компиляции библиотек компонентов.



■ *Рис. 1. Диаграмма библиотеки CADLib*

Основы использования библиотеки CADLib

Класс CAComponent. Как отмечалось выше, данный класс является базовым для всех компонентов. Опишем его члены.

- public CAComponent() – конструктор;
- public ~CAComponent() – деструктор;
- protected int m_iType – переменная, хранящая тип компонента. Может принимать одно из следующих значений:

CT_COMPONENT – абстрактный тип, среда игнорирует такой компонент;
CT_GRID – решетка;
CT_DATUM – хранилище данных;
CT_METRICS – метрика;
CT_RULES – правила;
CT_ANALYZER – анализатор.
Значение присваивается в потомках.

- public inline UINT GetType() – функция возвращает тип компонента.

Далее описываются функции, предоставляющие информацию о компоненте. Их можно или нужно переопределять в наследниках.

- public virtual inline STRING GetName() – функция возвращает имя компонента. Она должна быть переопределена в потомке. Для переопределения предусмотрен макрос COMPONENT_NAME(<имя>).
- public virtual inline STRING GetInfo() – функция возвращает описание компонента. Она должна быть переопределена в потомке. Для переопределения предусмотрен макрос COMPONENT_INFO(<информация>).
- public virtual inline LPTSTR GetIconID() – функция возвращает идентификатор ресурса-иконки данного компонента. Если она возвращает NULL, то среда будет использовать иконку по умолчанию. Функция может быть переопределена в потомке, по умолчанию возвращает NULL. Для переопределения предусмотрен макрос COMPONENT_ICON(<идентификатор>).
- public virtual inline STRING GetRequire() – функция возвращает выражение, описывающее набор свойств, которыми должны обладать компоненты для того, чтобы быть совместимыми с данным (так называемые «условия совместимости»). Выражение принадлежит языку, определяемому следующей грамматикой:

```
<EXPRESSION> ::= (<EXPRESSION>)
<EXPRESSION> ::= !<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>^<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>|<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>&<EXPRESSION>
<EXPRESSION> ::= <TOKEN>
```

Каждая лексема <TOKEN> может представлять собой:

конкретное свойство, набор слов, разделенных точкой (например, «Data.bool» или «Metrics.2D.cartesian»). Каждое следующее слово уточняет предыдущее или, иными словами, является «подсвойством» предшествующего;

начало свойства и групповой символ «*» (например, «Metrics.2D.*»), обозначающий любое, в том числе и нулевое количество подсвойств;

начало свойства и групповой символ «?», обозначающий любое ненулевое количество подсвойств.

В качестве имен свойств и подсвойств могут быть использованы произвольные слова. При их проверке на совпадение регистр не учитывается.

Функция может быть переопределена в потомке, по умолчанию возвращает «*», что соответствует совместимости со всеми компонентами. Для переопределения предусмотрен макрос COMPONENT_REQUIRES(<выражение>).

- public virtual inline STRING GetRealize() – функция возвращает набор конкретных свойств, которые реализует настоящий компонент. Если свойств несколько, то они разделяются точкой с запятой. Функция может быть переопределена в потомке, по умолчанию возвращает пустую строку, что приводит к тому, что данный компонент будет совместим со всеми другими. Для переопределения предусмотрен макрос COMPONENT_REALIZES(<выражение>).

Далее описываются функции для поддержки работы с параметрами, наследниками класса Parameter. Средой обеспечивается возможность изменения их значений для настройки компонента.

- public virtual inline unsigned short GetParametersCount() – функция возвращает число параметров данного компонента. Она может быть переопределена в потомке, по умолчанию возвращает ноль.
- public virtual inline Parameter* GetParameter(unsigned short n) – возвращает указатель на *n*-й параметр компонента, где *n* принимает значение от нуля до значения, которое вернет функция GetParametersCount, уменьшенного на единицу. Функция может быть переопределена в потомке, по умолчанию возвращает NULL.

Для удобства реализации двух последних функций предусмотрены следующие макроопределения:

PARAMETERS_COUNT(<число_параметров>) для задания числа параметров;

BEGIN_PARAMETERS для того, чтобы начать так называемую «карту параметров»;

PARAMETER(n,<указатель_на_n-ый_параметр>) для добавления *n*-го параметра в карту;

END_PARAMETERS для того, чтобы закончить карту.

Например, если у компонента *N* параметров, носящих имена p_Par1, p_Par2, ..., p_ParN, то карта параметров будет иметь вид:

```
PARAMETERS_COUNT(N)
BEGIN_PARAMETERS
    PARAMETER(0,&p_Par1)
```

```
    PARAMETER(1,&p_Par2)
            ...
    PARAMETER(N-1,&p_ParN)
END_PARAMETERS
```

• public virtual void ParameterChanged(Parameter* pPar) – функция, вызываемая ядром, когда в среде произошло изменение значения некого параметра данного компонента. При этом указатель на изменившийся параметр передается в функцию. При инициализации компонента ей передается значение NULL. Функция может быть переопределена в потомке, по умолчанию ничего не делает.

Класс CARules. Данный класс является базовым для всех компонентов, реализующих правила клеточного автомата. Класс обладает широким спектром возможностей. Он включает в себя, в частности, средства для межавтоматных взаимодействий, кластерных вычислений и т. д. Опишем здесь только основные его члены:

• public CARules() – конструктор.

• protected CAUnion* m_pUnion – переменная хранит указатель на «союз компонентов», с которыми должны работать данные правила автомата. Союзом компонентов называется набор из совместимых друг с другом решетки, метрики и хранилища данных.

• public void SetUnion(CAUnion* pUnion) – функция устанавливает значение указателя на союз компонентов.

• public CAUnion* GetUnion() – функция возвращает значение указателя на союз компонентов.

• public EnvInfo* m_pEnvInfo – переменная хранит объект класса EnvInfo, служащего для описания платформы и окружения, в котором выполняются вычисления. Эта информация может быть проанализирована и учтена функциями класса CARules.

• public void SetEnvInfo(EnvInfo* pEnvInfo) – функция устанавливает описания платформы и окружения.

• public bool m_bFinalizeIfChanged – если значение этой переменной равно true, то после любого изменения состояния решетки пользователем будет произведена финализация эксперимента и перед следующим шагом потребуется повторная инициализация. Эта возможность оказывается полезной при использовании средств оптимизации производительности, кластерных вычислениях и т. д.

Далее описываются функции для управления итерациями, которые могут быть переопределены в потомках.

• public virtual bool SubCompute(Zone& z) – функция выполняет шаг автомата в области решетки, ограниченной значением первой координаты от z.a1 до z.b1, второй – от z.a2 до z.b2 и третьей – от z.a3 до z.b3. Эта функция является вспомогательной. Она вызывается функцией Compute при распараллеливании вычислений на многопроцессорной системе, например, с помощью вспомогательного класса RulesThread, а также при использовании кластеров.

Необходимо отметить, что идеологически правильно применять ее и в тех случаях, когда вычисления не распределяются.

Клеточные автоматы характеризуются тем, что результирующее состояние решетки не зависит от порядка обхода клеток или областей. При выполнении этого свойства вызов данной функции для множества непересекающихся зон, объединение которых покрывает всю решетку, приведет к тому же результату, что и вызов ее для единственной области, охватывающей всю решетку целиком.

Функция может быть переопределена в потомке, по умолчанию возвращает true.

Необходимо отметить, что, например, при использовании обобщенных координат [7, 18] даже для двумерных и трехмерных решеток область ограничивается только вдоль одной оси.

• public bool SubCompute() – функция вызывает предыдущую функцию, передавая в качестве параметра описание всей зоны, охватываемой текущим хранилищем данных.

• public virtual bool Initialize() – функция выполняет инициализацию эксперимента перед его стартом или выполнением шага после финализации. Если функция возвращает true, то инициализация считается прошедшей успешно и эксперимент может быть начат. Она может быть переопределена в потомке, по умолчанию возвращает true.

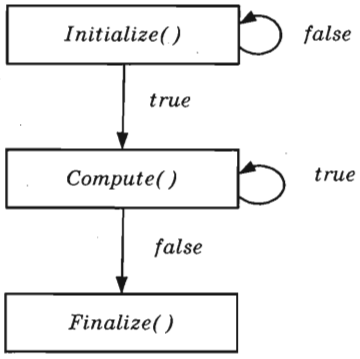
• public virtual bool Compute() – функция выполняет шаг автомата. Если функция возвращает true, то эксперимент может быть продолжен после данного шага. В противном случае он будет закончен и финализирован. Таким образом, можно задавать критерии окончания эксперимента. Функция может быть переопределена в потомке, по умолчанию вызывает функцию SubCompute, передавая в качестве параметра описание всей зоны, охватываемой текущим хранилищем данных.

• public virtual void Finalize() – функция выполняет финализацию эксперимента. Она может быть переопределена в потомке, по умолчанию не делает ничего.

Порядок вызова последних трех функций при проведении эксперимента иллюстрирует рис. 2.

Кроме того, для управления состоянием решетки в классе предусмотрены следующие функции, которые могут быть переопределены в потомке.

• public virtual void Tool1(), public virtual void Tool2(), public virtual void Tool3() – функции реализуют три «пользовательских инструмента», которые могут быть вызваны из среды. Эти инструменты позволяют преобразовывать или анализировать состояние решетки. По



■ *Рис. 2. Схема вызова функций класса* CARules *при проведении эксперимента*

умолчанию функции отображают сообщения о том, что они не были переопределены.

В случае, если пользователю потребуется больше трех инструментов, можно использовать параметры-инструменты (объекты класса ParamTool), реализуя их внутри функции ParameterChanged.

Для определения анализируемых параметров, характеризующих эксперимент, предусмотрены две функции:

• public virtual inline unsigned short GetAnalyzableParametersCount() – функция возвращает число анализируемых параметров данного компонента правил. Она может быть переопределена в потомке, по умолчанию возвращает ноль.

• public virtual inline Parameter* GetAnalyzableParameter(unsigned short n) – возвращает указатель на n-й анализируемый параметр компонента, где n принимает значение от нуля до значения, которое вернет функция GetAnalyzableParametersCount, уменьшенного на единицу. Функция может быть переопределена в потомке, по умолчанию возвращает NULL.

Для удобства реализации двух последних функций, как и в случае обычных параметров, предусмотрены макроопределения:

APARAMETERS_COUNT(<число_параметров>) для задания числа анализируемых параметров;

BEGIN_APARAMETERS для того, чтобы начать карту анализируемых параметров;

END_APARAMETERS для того, чтобы закончить карту.

Параметры в карту добавляются с помощью описанного выше макроопределения PARAMETER.

Остальные функции и переменные, члены класса CARules, останутся за пределами рассмотрения в настоящей работе, так их общее количество слишком велико. Отметим лишь, что некоторые из них будут обсуждаться в контексте организации кластерных вычислений.

Базовые сведения о параметрах. В библиотеку CADLib входит набор классов, реализующих параметры различных типов. Как отмечалось выше, все они являются потомками класса Parameter. Однако у них есть еще один общий предок, шаблон TypedParameter, непосредственный наследник класса Parameter. Этот шаблон, помимо прочего, реализует две следующие функции (будем считать type аргументом шаблона, определяющим тип параметра):

• virtual inline type Get() – функция возвращает значение параметра;

• virtual inline bool Set(type value, bool notify=true) – функция устанавливает значение параметра равным value. Если флаг notify равен true, то при этом будет вызвана функция ParameterChanged компонента, владеющего данным параметром, иначе – не будет. Функция возвращает true, если новое значение корректно, и оно было присвоено. В противном случае она возвращает false. Функция должна быть переопределена в потомке.

В библиотеку CADLib также входит набор классов, реализующих окна-диалоги для запроса значений параметров всех предусмотренных типов из среды.

В таблице приведены имена классов, реализующих стандартные параметры, имена классов соответствующих им диалогов, а также словесные описания типов параметров.

Необходимо также отметить, что у класса каждого параметра имеется конструктор, который в качестве аргументов принимает имя параметра, его описание, начальное значение и указатель на компонент, которому принадлежит данный параметр. При этом конструктора по умолчанию у параметров нет, поэтому они должны создаваться в конструкторе владеющего ими компонента.

Разработка библиотеки, содержащей компонент. Как отмечалось выше, компонент должен быть помещен в динамическую библиотеку DLL (Dynamic-Link Library). Помимо класса, реализующего компонент, в ней должны находиться три функции.

1. EXPORTIT[1] STRING[2] GetCompatibilityInfo() – функция аутентификации библиотеки и компонента. Возвращает строку вида «CAMEL Component| <версия_ядра>|<тип_компонента>», где

<версия_ядра> – версия ядра программного обеспечения, с которой совместим данный компонент, например «1.0». Совместимость «вниз», по возможности, должна обеспечиваться;

<тип_компонента> – строка «Grids», «Data», «Metrics», «Rules» или «Analyzers», в зависимости от типа компонента, содержащегося в библиотеке.

[1] Использование макроопределения EXPORTIT обеспечивает экспортирование функции из библиотеки.

[2] Макроопределение STRING обозначает указатель на первый символ строки с завершающим нулем.

■ *Стандартные классы параметров, соответствующие им классы диалогов и описания их типов*

Класс параметра	Класс диалога	Описание типа параметра
ParamInt	CIntDlg	Целое число
ParamDouble	CDoubleDlg	Число с плавающей точкой двойной точности
ParamBool	CBoolDlg	Булева величина
ParamColor	CColorDlg	Дескриптор цвета
ParamIntGap	CIntGapDlg	Отрезок с целочисленными границами
ParamChoice	CChoiceDlg	Значение из конечного множества вариантов
ParamContSpectrum	CContSpectrumDlg	Непрерывный цветовой спектр
ParamDiscSpectrum	CDiscSpectrumDlg	Дискретный цветовой спектр
ParamCell	CCellDlg	Идентификатор клетки решетки
ParamString	CStringDlg	Строка
ParamFilePath	CFilePathDlg	Путь к файлу
ParamTool	CToolDlg	Инструмент

Для удобства определения этой функции предусмотрены макроопределения COMPATIBLE_GRID(<версия_ядра>), COMPATIBLE_METRICS(<версия_ ядра>), COMPATIBLE_DATUM(<версия_ядра>), COMPATIBLE_RULES(<версия_ ядра>) и COMPATIBLE_ANALYZER(<версия_ядра>).

2. Функция создания компонента, возвращающая указатель на новый экземпляр. В зависимости от типа она может иметь вид:

```
EXPORTIT CAGrid* CreateGrid();
EXPORTIT CAMetrics* CreateMetrics();
EXPORTIT CADatum* CreateDatum();
EXPORTIT CARules* CreateRules();
EXPORTIT CAAnalyzer* CreateAnalyzer().
```

3. Функция удаления компонента, освобождающая память, занимаемую им. В зависимости от типа функция удаления может иметь вид:

```
EXPORTIT DestroyGrid(CAGrid* pGrid);
EXPORTIT DestroyMetrics(CAMetrics* pMetrics);
EXPORTIT DestroyDatum(CADatum* pDatum);
EXPORTIT DestroyRules(CARules* pRules);
EXPORTIT DestroyAnalyzer(CAAnalyzer* pAnalyzer).
```

Для удобства реализации двух последних функций предусмотрены макроопределения GRID_COMPONENT(<имя_класса>), METRICS_COMPONENT(<имя_класса>), DATUM_COMPONENT(<имя_класса>), RULES_COMPONENT(<имя_класса>) и ANALYZER_COMPONENT(<имя_класса>).

В результате, например, при написании библиотеки, содержащей класс компонента решетки CAMyGrid, совместимого с ядром версии 1.0, помимо реализации самого класса достаточно добавить две строки:

```
COMPATIBLE_GRID(1.0)
GRID_COMPONENT(CAMyGrid)
```

Для написания компонентов рекомендуется использовать компилятор и среду разработки Microsoft Visual C++ 6, так как пакет CAME&L создавался с помощью этого средства. Таким образом, весь набор примеров, стандартных компонентов, также ориентирован на него.

Для реализации нового компонента следует создать в среде разработки проект, представляющий собой динамическую библиотеку (выбрать в меню «File» | «New...» | «Projects» | «Win32 Dynamic-Link Library», указать имя и директорию). В появившемся после этого мастере выбрать вариант «An empty DLL project». После этого необходимо подключить к проекту библиотеку CADLib (выбрать в меню «Project» | «Settings» | «Link» и в категории «General» в поле «Object/library modules» добавить «CADLib.lib», а в категории «Input» в поле «Additional library path» добавить «%CAMEL_PATH%[1]\Lib»). Теперь можно приступать к разработке, создавать новые файлы, реализующие компонент, а также его иконку.

Однако для того, чтобы избавить пользователя от необходимости настраивать среду для каждого нового компонента, был разработан скрипт ccomp.sh, служащий для копирования существующих компонентов. Под операционной системой

[1] Будем обозначать через %CAMEL_PATH% путь к установленному пакету CAME&L.

семейства Windows он может быть выполнен с помощью пакета Cygwin [20], средства эмуляции оболочки операционных систем семейства UNIX/Linux.

Данный скрипт доступен с сайта [7]. Перед использованием его необходимо поместить в директорию «%CAMEL_PATH%\CompsSrc\ <тип_компонента>», в которой обычно размещаются исходные коды компонентов.

Скрипт поддерживает следующие параметры из командной строки:

```
ccomp.sh <директория_из_которой_копировать> <директория_в_которую_ копировать> <опции>
```

Поддерживаются следующие опции (в алфавитном порядке):

`-c <имя_класса>` – установить соответствующее имя класса для копии компонента;

`-i <описание>` – установить соответствующее описание для копии компонента;

`-n <имя>` – установить соответствующее имя[1] для копии компонента.

Для получения информации о параметрах командной строки можно также воспользоваться встроенной в скрипт помощью, вызвав

```
ccomp.sh -h
```

Например, если в директории %CAMEL_PATH%\CompsSrc\Rules\MyRules1 находится компонент, который можно взять за основу нового компонента правил, то следует воспользоваться описанным выше скриптом, поместив его в директорию %CAMEL_PATH%\CompsSrc\Rules и вызвав

```
ccomp.sh MyRules1/ MyRules2/ -c CAMyRules2 -i "This is my second rules component" -n MyRules2
```

В результате компонент будет скопирован в директорию %CAMEL_PATH%\CompsSrc\Rules\MyRules2. В копии будет установлено имя класса `CAMyRules2`, описание «`This is my second rules component`» и имя компонента «`MyRules2`».

Описанный выше скрипт для копирования компонентов позволяет существенно увеличить скорость разработки новых решений.

Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту № 05-07-90086 «Разработка среды и библиотеки "CAME&L" для организации параллельных и распределенных вычислений на основе клеточных автоматов».

Литература

1. **MacDonald N., Minty E., Harding T., Brown S.** Writing Message Passing Parallel Programs with MPI – Edinburgh Parallel Computing Center. – The University of Edinburgh, 1996.
2. **Geist A., Beguelin A., Dongarra J.** and oth. PVM: Parallel Virtual Machine. A User Guide and Tutorial for Networked Parallel Computing. – MIT Press, Cambridge, 1994.
3. **Foster I., Kesselman C., Tuecke S.** The Anatomy of Grid // http://www.globus.org, 2001.
4. **Foster I., Kesselman C., Nick J. M., Tuecke S.** The Physiology of Grid // http://www.globus.org, 2002.
5. **Foster I.** What is a Grid? A Three Point Checklist // http://www.globus.org, 2002.
6. **Foster I., Kesselman C.** Globus: A Metacomputing Infrastructure Toolkit // http://www.globus.org, 2002.
7. **Сайт** «CAMEL Laboratory» – http://camellab.spb.ru.
8. **Naumov L.** CAME&L – Cellular Automata Modeling Environment & Library // Cellular Automata. Sixth International Conference on Cellular Automata for Research and Industry, ACRI-2004. – Springer–Verlag, 2004.
9. **Наумов Л.** CAME&L – Среда моделирования и библиотека разработчика клеточных автоматов // Труды XI Всероссийск. научно-метод. Конф. Телематика'2004. – Т. 1. – СПб.: СПбГУ; ИТМО, 2004.
10. **фон Нейман Дж.** Теория самовоспроизводящихся автоматов. – М.: Мир, 1971.
11. **Тоффоли Т., Марголус Н.** Машины клеточных автоматов. – М.: Мир, 1991.
12. **Wolfram S.** A New Kind of Science. – IL: Wolfram Media, 2002.
13. **Sloot P.M.A., Chopard B., Hoekstra A.** Cellular Automata. Sixth International Conference on Cellular Automata for Research and Industry, ACRI-2004. – Springer-Verlagю, 2004.
14. **Sarkar P.** A Brief History of Cellular Automata // ACM Computing Surveys. – Vol. 32. –№ 1. – 2000.
15. **Наумов Л.** CTP (Commands Transfer Protocol) – Сетевой протокол для высокопроизводительных вычислений // Труды XI Всероссийск. научно-метод. конф. Телематика'2004. – Т. 1. – СПб.: СПбГУ; ИТМО, 2004.
16. **Naumov L.** Commands Transfer Protocol (CTP) – New Networking Protocol for Parallel or Distributed Computations // http://www.codeproject.com/internet/ctp.asp, 2004.
17. **Страуструп Б.** Язык программирования C++. – 3-е изд. СПб.: Невский диалект, 1999.
18. **Naumov L.** Generalized Coordinates for Cellular Automata Grids // Computational Science – ICCS 2003. – P. 2. – Springer-Verlag, 2003.
19. **Мешков А., Тихомиров Ю.** Visual C++ и MFC. – 2-е изд. – СПб.: БХВ–Санкт-Петербург, 2000.
20. http://www.cygwin.com.

[1] Здесь имеется в виду имя, под которым компонент виден в среде, возвращаемое его функцией `GetName()`, а не имя класса.

УДК 519.872; 519.876.5

РАСЧЕТ МНОГОУРОВНЕВОЙ СИСТЕМЫ ОЧЕРЕДЕЙ

Ю. И. Рыжиков,
доктор техн. наук, профессор
Военно-космическая академия им. А. Ф. Можайского

Излагаются соображения в пользу коллективного использования вычислительных ресурсов. Предложен основанный на законе сохранения объема работы метод расчета среднего времени пребывания заявок в одноканальной многоуровневой системе с квантованным обслуживанием в зависимости от заявленного времени счета. Длительности квантов и величина системных потерь предполагаются фиксированными и переменными по уровням.

The arguments are discussed for collective using of computer resources. An algorithm is proposed to compute mean sojourn time depending on requested computing time. The system is supposed to be one-channel and multi-level with quantified servicing, quanta and system losses being fixed and varying by levels.

Одной из ведущих тенденций в современной информатике является коллективное использование имеющихся вычислительных ресурсов. В сравнении с монопольным доступом такой подход дает следующие преимущества:

1. Клиент системы платит лишь за фактически использованные ресурсы.

2. В связи с относительным удешевлением единицы продукции на более мощных установках эти ресурсы обходятся дешевле (согласно закону Гроша стоимость вычислительной установки растет пропорционально квадратному корню из ее производительности).

3. Улучшаются показатели обслуживания за счет масштабного эффекта (n-кратное увеличение интенсивности потока заявок и обслуживания во столько же раз уменьшает среднее время ожидания заявки – этот неожиданный факт элементарно доказывается для системы M/M/1, но имеет место и в более общих ситуациях).

4. Уменьшается доля необходимых страховых ресурсов мощностей.

5. На более производительных установках поднимается потолок пиковых потребностей и появляется возможность оказывать качественно новые виды услуг.

Идея коллективного использования ресурсов по фактической потребности при сохранении психологических преимуществ индивидуального доступа еще в 1960-х гг. нашла свое отражение в операционных системах мощных стационарных ЭВМ (mainframes). В наши дни те же проблемы возникают вновь на более высоком уровне при использовании суперкомпьютеров (М-1000 в ВЦ РАН); при организации распределенных (кластерных) вычислений; при работе в локальных сетях с «тонкими» клиентами, переадресующими все серьезные вычисления серверу; при реализации оперативных задач на фоне трудоемких массовых вычислений (например, в вузовских ВЦ); при работе сетевых маршрутизаторов и т. д.

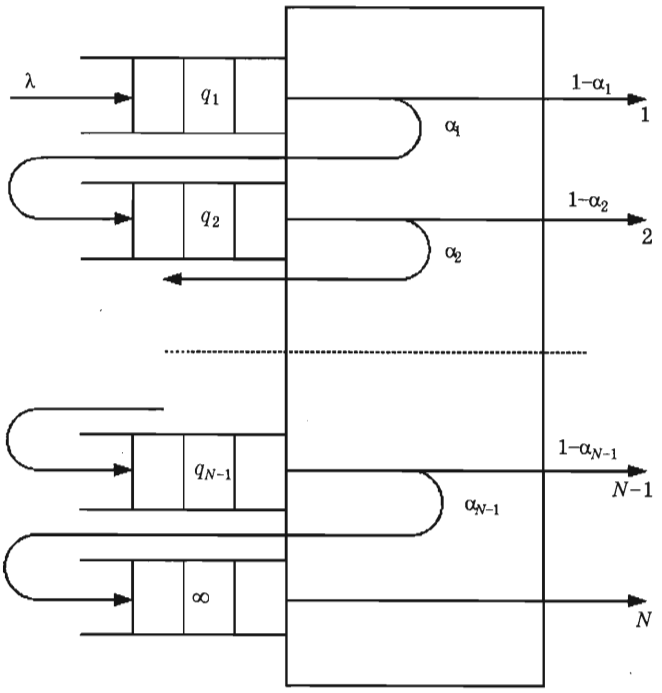
Ключевым элементом технологии коллективного использования процессора является *квантованное* обслуживание, при котором каждая из находящихся в системе активных задач поочередно получает «квант» времени процессора. В зависимости от порядка его предоставления рассматриваются различные дисциплины обслуживания:

циклические RR (Round Robin – «карусель»);

многоуровневые с N очередями разного приоритета FB_N (foreground-background, т. е. с передним и задним планом);

уравнительное разделение процессора EPS (egalitarian processor sharing).

Важнейшим показателем эффективности математической эксплуатации вычислительной системы коллективного доступа является количество задач, решенных на ней в единицу времени. Этот показатель будет максимален, если предоставить статический приоритет коротким заявкам. Однако в типичных условиях случайной трудоемкости эта стратегия физически не реализуема, поскольку длины заявок априорно не известны. Все перечисленные выше дисциплины *автоматически* обеспечивают приоритет корот-



■ *Расчет многоуровневой системы*

ким заявкам и к тому же по фактической, а не по ожидаемой длительности. Естественно, что делается это за счет длинных заявок и ценой дополнительных системных издержек на прерывания и обмены. Поэтому при анализе упомянутых дисциплин показатели обслуживания определяются в зависимости от длины заявки t. Основным показателем является среднее время ν_t ее пребывания в системе или среднее время ν_k пребывания в системе заявки, завершающей обслуживание на k-м кванте.

Затраты τ времени работы процессора на каждое переключение увеличивают общую загрузку системы циклического обслуживания и особенно сильно сказываются на прохождении «длинных», много раз прерываемых задач. Многоуровневая система [1–4, 6, 7] типа показанной на рисунке дает наибольшие преимущества коротким заявкам и к тому же уменьшает частоту переключений. Эта дисциплина в работе [7] называется Shortest Elapsed Time (SPT). В такой системе заявка, не завершившая обслуживание на i-м уровне в течение кванта q_i, переходит на следующий уровень и получает на нем квант $q_{i+1} > q_i$. Нижний уровень N предоставляет обслуживание до его завершения. Обслуживание заявок из разных очередей производится по схеме относительного приоритета. Переход к каждой новой заявке требует времени процессора τ независимо от уровня обслуживания.

Во всех перечисленных источниках поставленная задача решается при существенных ограничениях. В работах [1, 3] общая длительность обслуживания предполагается показательно распределенной, в работе [3] то же допущение делается относительно длительности кванта; в работах [1, 7] кванты считаются постоянными и не зависящими от номера уровня, а в исследовании [2] – бесконечно малыми. Описываемые методы, как правило, неконструктивны: к примеру, в работе [2] предлагается решать функциональное уравнение в преобразованиях Лапласа, а в работе [6] бесконечную систему линейных уравнений, причем среднее время ожидания на j-м уровне зависит от таковых на последующих уровнях (это явная ошибка). Нигде не учитываются играющие принципиальную роль потери на переключения и не приводится верификация расчетных методик.

Опишем свободный от указанных недостатков метод расчета среднего времени ν_k пребывания в одноканальной системе заявки, обслуживание которой завершается на k-м уровне, $k = 1, N$. Для всех уровней введем понятия:

$Q_k = \sum_{i=1}^{k} q_{i,1}$ – интегральный квант;

$L_k = \sum_{i=1}^{k} \tau_{i,1}$ – средние интегральные потери до k-го уровня включительно;

$\alpha_k = \overline{B}(Q_k)$ – вероятность перехода заявки на следующий уровень ($\alpha_0 = 1$);

$b_{k,m}^{(p)} = \int_{Q_{k-1}}^{Q_k} t^m b(t)dt$ – частичный момент трудоемкости порядка m;

$b_{k,m}^{(o)} = \int_{Q_{k-1}}^{Q_k} (t - Q_{k-1})^m b(t)dt$ – момент длительности неполного кванта;

$b_{k,m}^{(W)} = \alpha_k q_{k,m} + b_{k,m}^{(o)}$ – «взвешенный» момент длительности k-обслуживания, учитывающий вероятную потребность в полном кванте;

$b_k^{(L)} = b_k^{(W)} + \tau_k$ – «нагрузочные» моменты, получаемые сверткой взвешенных моментов и системных потерь (распределение последних зависит от номера уровня, если информация по соответствующим заявкам размещается в запоминающих устройствах разных типов);

$\lambda_k = \lambda\alpha_{k-1}$ – интенсивность потока на k-м уровне;

w_k – среднее время ожидания обслуживания на k-м уровне;

W_k – среднее время ожидания в очередях до k-го уровня включительно.

Интересующий нас конечный результат

$$\upsilon_k = W_k + Q_{k-1} + L_k + b_{k,1}^{(o)}.$$

Ключевым элементом этой технологии является расчет средних времен $\{w_k\}$ ожидания меченой заявкой очередного кванта на k-м уровне для $k \geq 2$. Выделим моменты времени:

X – прибытия меченой заявки в систему,

Y – начала ее обслуживания на $(k-1)$-м уровне,

Z – ее перехода на уровень k.

Назовем заявку, обслуживаемую в момент X (эта заявка с вероятностью ρ_i выбирается из i-й очереди), A-заявкой.

Обслуживание меченой заявки на $(k-1)$-м уровне может начаться лишь при отсутствии заявок на всех вышележащих уровнях. К этому моменту (Y) все заявки, которые находились в системе в момент X, окажутся уже на уровне k, но в связи с наличием более приоритетной меченой обслуживаться пока не будут (здесь и далее имеется в виду та часть заявок, которая добралась до соответствующего уровня). Они создадут для меченой среднюю задержку T_1. Им будет предшествовать A-заявка (задержка T_2).

Все заявки, пришедшие после меченой, окажутся позади нее на уровне $(k-1)$ и после момента Z меченая заявка будет ждать прохождения ими этого уровня (задержка T_3). Наконец, за время обслуживания меченой заявки на $(k-1)$-м уровне в систему придут новые заявки, и меченая будет ждать прохождения ими уровней $i = \overline{1,\ k-1}$ – задержка T_4.

Запишем формулы подсчета средних значений упомянутых задержек:

$$T_1 = r_{k,\,k}\left[\overline{B}(Q_{k-1})\sum_{i=1}^{k-1}\lambda_i w_i + \lambda_k w_k\right];$$

$$T_2 = r_{k,\,k}\,\overline{B}(Q_{k-1})\sum_{i=1}^{k-1}\rho_i;$$

$$T_3 = \lambda r_{k-1,\,k-1}\,\overline{B}(Q_{k-2})\left[\sum_{i=1}^{k-2}(w_i + \tau_i + q_i) + w_{k-1}\right];$$

$$T_4 = \lambda r_{1,\,k-1}(\tau_{k-1} + q_{k-1}).$$

После приравнивания w_k их суммы и элементарных преобразований получаем для всех $k \geq 2$ формулу

$$w_k = E_k / F_k,$$

где

$$E_k = \overline{B}(Q_{k-1})r_{k,\,k}\left[\lambda\sum_{i=1}^{k-1}\overline{B}(Q_{i-1})w_i + \sum_{i=1}^{k-1}\rho_i\right] +$$

$$+ \lambda\left\{(\tau_{k-1} + q_{k-1})r_{1,k-1} + \overline{B}(Q_{k-2})\times\right.$$

$$\left.\times\left[\lambda\sum_{i=1}^{k-2}(w_i + \tau_i + q_i) + w_{k-1}\right]r_{k-1,\,k-1}\right\};$$

$$F_k = 1 - \lambda\left[r_{1,\,k-1} + \overline{B}(Q_{k-1})r_{k,\,k}\right].$$

Для первого уровня расчет выполняется как для обычных систем с относительным приоритетом.

Трудоемкости $\{r_{i,j}\}$ обработки заявки на уровнях $i = \overline{1,\ N}$ включительно с учетом отсева полностью обслуженных вычисляются рекуррентно:

$$r_{N,N} = \tau_N + \frac{1}{\overline{B}(Q_{N-1})}\int_{Q_{N-1}}^{\infty}(t - Q_{N-1})dB(t) =$$

$$= \tau_N + \frac{1}{\overline{B}(Q_{N-1})}\left[b_1 - \int_{Q_{N-1}}^{\infty} t\,dB(t) - Q_{N-1}\overline{B}(Q_{N-1})\right];$$

$$r_{i,N} = \tau_i + \frac{1}{\overline{B}(Q_{i-1})}\int_{Q_{i-1}}^{Q_i}(t - Q_{i-1})dB(t) + B(Q_i)\left[q_i + r_{i+1,N}\right],$$

$$i = N-1,\ N-2,\ \ldots,\ 1.$$

Для ярусов $j = \overline{1,\ N-1}$ аналогичные формулы имеют вид

■ ***Средние времена ожидания по уровням***

Уровни	λ = 0,5		λ = 0,6	
	Имитация	Расчет	Имитация	Расчет
1	0,222	0,222	0,275	0,276
2	0,289	0,287	0,429	0,427
3	1,382	1,328	2,506	2,411
4	3,669	3,384	8,541	7,922
5	4,757	4,623	11,459	11,489

$$r_{j,j} = \tau_j + \frac{1}{\overline{B}(Q_{j-1})}\left[\int_{Q_{j-1}}^{Q_j}\left(t - Q_{j-1}\right)dB(t) + \overline{B}(Q_j)q_j\right];$$

$$r_{i,j} = \tau_i + \frac{1}{\overline{B}(Q_{i-1})}\times$$

$$\times\left\{\int_{Q_{i-1}}^{Q_i}\left(t - Q_{i-1}\right)dB(t) + \overline{B}(Q_i)\left[q_i + r_{i+1,j}\right]\right\},$$

$$i = j-1,\ j-2,\ \ldots,\ 1.$$

Для завершения описания методики остается обсудить технику расчета частичных моментов. Эти моменты, если соответствующие интегралы явно не берутся, удобно считать численным интегрированием по Симпсону с последовательным делением шага пополам и вычислением только дополнительных ординат [4].

Возможно также (после аппроксимации $b(t)$ гамма-плотностью с поправочным многочленом) использование известных степенных рядов и рекуррентных соотношений для неполной гамма-функции [5]. К несложным формулам приводит и H_2 -аппроксимация распределения времени обслуживания.

Для последнего уровня $Q_N = \infty$, и частичные моменты получаются вычитанием ранее полученных из полных:

$$b_{N,m}^{(p)} = b_m^{(p)} - \sum_{i=1}^{N-1} b_{i,m}^{(p)}.$$

Описанная методика была запрограммирована и реализована применительно к пятиуровневой системе с начальным квантом $q_1 = 0{,}2$, удвоением кванта на уровнях до 4-го включительно, при постоянных потерях на переключение $\tau = 0{,}1$ и длительности обслуживания по закону Эрланга 2-го порядка с единичным средним. Результаты представлены в таблице.

Таким образом, расчет вполне согласуется с имитационным экспериментом.

Многоуровневая система несколько ухудшает оперативность обслуживания коротких заявок, которые могут застать ЦП занятым выдачей длинного кванта. Поэтому имеет смысл наиболее длинным заявкам выдавать требуемый квант по частям, в интервалах между которыми возможно обслуживание коротких заявок. Соответственно последний уровень на рисунке разбивается на несколько подуровней. Такая схема может быть применена для анализа квантованного приоритетного обслуживания, реализуемого на фоне вычислений большой трудоемкости.

При малых «дробных» квантах она фактически предоставляет коротким заявкам приоритет с прерыванием, хотя и несколько задержанным.

Укажем еще несколько возможных обобщений многоуровневой схемы:

прямой доступ заявок из внешнего источника на промежуточные уровни;

обслуживание части верхних уровней с абсолютным приоритетом (в этом случае к системе в целом применяется алгоритм расчета модели со смешанными приоритетами);

выдача «дробных» квантов случайной длительности с объединением подуровней в одну RR-систему (этот вариант хорошо моделирует длительный счет в фоновом разделе, прерываемый в моменты промежуточных выдач);

распределение недообслуженных заявок с определенными вероятностями между несколькими нижележащими уровнями.

Комбинация двух последних вариантов особенно удачно описывает реальные процессы обработки неоднородных заявок (на первом уровне вновь прибывшие заявки классифицируются по системным очередям). Разумеется, все перечисленные в начале статьи сферы коллективного использования вычислительных ресурсов имеют заметную специфику, но описанная схема может служить источником полезных аналогий и технологических рецептов.

В заключение отметим ее педагогическую ценность при изучении дисциплин типа «Компьютерное моделирование», поскольку составление уравнений баланса объемов работы на каждом из уровней далеко не тривиально и способствует лучшему уяснению законов сохранения теории очередей [4].

Литература

1. **Балыбердин В. А.** Методы анализа мультипрограммных систем. – М.: Радио и связь, 1992. – 152 с.
2. **Клейнрок Л.** Вычислительные системы с очередями / Пер. с англ. – М.: Мир, 1979. – 600 с.
3. Основы теории вычислительных систем /Под ред. С. А. Майорова. – М.: Высшая школа, 1978. – 408 с.
4. **Рыжиков Ю. И.** Машинные методы расчета систем массового обслуживания. – Л.: ВИКИ им. А. Ф. Можайского, 1979. – 177 с.
5. **Янке Е., Эмде Ф., Леш Ф.** Специальные функции. - М.: Наука, 1968. – 344 с.
6. **Яшков С. Ф.** Анализ очередей в ЭВМ. – М.: Радио и связь, 1989. – 216 с.
7. **Coffman E. G., Denning P. J.** Operating Systems Theory. – Englewood Cliffs, NJ: Prentice-Hall, 1973. – 331 p.

УДК 621.856

СИСТЕМНЫЙ АНАЛИЗ И ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ КАК СРЕДСТВО ОБЪЕДИНЕНИЯ РЕЗУЛЬТАТОВ ГУМАНИТАРНЫХ И ТОЧНЫХ НАУК

Е. И. Перовская,
доктор техн. наук, профессор
Санкт-Петербургский государственный университет аэрокосмического приборостроения

Основной задачей данной работы является построение математической теории для описания метасистемы объединения разнородных моделей компонент социумов, являющихся результатами исследований как гуманитарных, так и точных наук. Разработанная теория и формальные методы предназначены для междисциплинарных исследований, учитывающих влияние результатов различных аспектов жизнедеятельности общества и природы как на отдельные социумы, так и на цивилизацию в целом при решении проблемы жизнеспособного (устойчивого) развития. Первая часть статьи содержит постановку проблемы создания имитационных моделей систем Человек–Культура–Технологии–Природа.

The work is directed on a solution of a fundamental problem of creation of global community models for checking of administrative solutions and their consequences without experimenting on Human and Nature. The basic goal is development of the mathematical theory of association of diverse community components models, which are researches results of the both humanitarian and exact sciences in uniform system. The developed theory and formal methods are intended for interdisciplinary researches which take into account influence of results of various aspects of functioning of a society and a nature, both on separate communities, and on a civilization as a whole at the decision of a problem of viable (sustainable) development. The first part of article contains statement of a problem of Human-Culture-Technology-Nature system simulation.

Системы и системный анализ

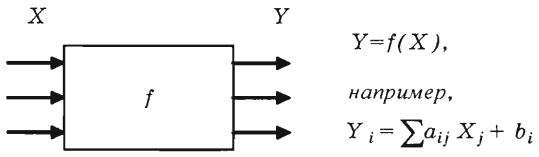
Несколько слов о терминах, которые будут использованы в этой статье. Не будем исследовать все многообразие смыслов, которые вкладываются в понятия «система», «модель», «системный анализ». Если заглянуть в Интернет, то по каждому из этих понятий можно найти тысячи статей. Споры идут не одно десятилетие, но это, видимо, и показывает, что понятие, отражающее какие-то свойства «чего-либо», очень сильно зависит от того, кто и что именно хочет сказать об этом «что-либо». Не будем спорить, просто уточним, что именно под каждым термином будем понимать в данном контексте. Начнем с термина, отражающего любое представление человека о чем бы то ни было – «модель».

Под термином «*модель*» будем понимать любое представление объекта (процесса или системы) в форме, отличной от формы его существования, отражающего его основные свойства, необходимые для решения конкретной задачи. Каждая модель адекватна только задаче, поставленной относительно моделируемой системы.

Системой будем называть совокупность компонент, объединенных между собой для получения определенных свойств и достижения целей, которые не могут быть достигнуты ни одной компонентой или неполной их совокупностью.

Под *компонентой* системы будем понимать любую часть системы, обладающую собственными целями и свойствами, необходимыми системе для достижения ее целей. Это означает, что в любой системе могут возникать противоречия между целями системы и целями ее компонент. Когда компонента может (или считает, что может) достигнуть своей цели самостоятельно, она выходит из системы, и ... так распадаются империи.

Элементами системы назовем неделимые части системы или компоненты, не имеющие собствен-



■ *Рис. 1. Структура задачи «черный ящик»*

ных целей, но обладающие свойствами, необходимыми системе для достижения ее целей.

В процессе формализации можно будет уточнять свойства введенных понятий, в соответствии с возникающими задачами или целями.

Несколько слов о системном анализе. На многочисленных конференциях по различным предметным и проблемным областям часто встречаются доклады с названием «Системный подход...» или «Системный анализ...», но редко можно получить ответ на вопрос, чем отличается системный подход от комплексного или системный анализ от анализа. В данной работе исходить будем из типа задач. Все задачи, возникающие при исследованиях, проектировании, диагностике, идентификации объектов, процессов или систем можно поделить на два класса по их целям.

К первому классу относятся задачи типа «черного ящика», когда задано (или наблюдено) множество значений входов X и выходов Y и требуется узнать закон функционирования объекта, т. е. функцию f, с помощью которой выходы объекта выражаются через входы $Y=f(X)$ (рис. 1).

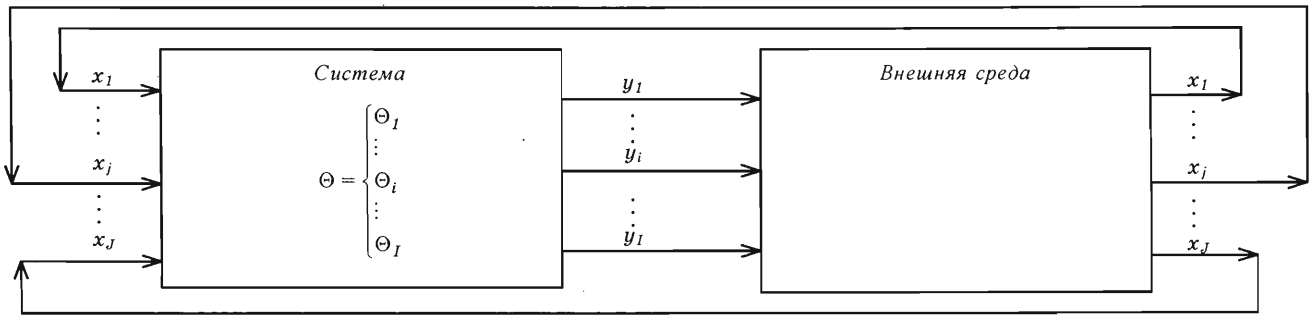
Ко второму классу относятся задачи, в которых интерес представляют влияние результатов функционирования исследуемой системы на внешнюю среду и влияние результатов жизнедеятельности внешней среды на исследуемую систему, т. е. все выходы компонент должны иметь компоненты-приемники и все входы должны иметь компоненты-источники. Рассматриваемая система взаимосвязанных компонент и внешней среды должна быть замкнутой (рис. 2).

Этот класс задач связан с появлением сложных многосвязных систем с компонентами различной природы, с технологиями функционирования компонент, способными уничтожить другие компоненты системы или внешнюю среду. Поэтому нельзя не учитывать связи, которые могут показаться слабыми, но при замыкании через нелинейные компоненты могут вызвать катастрофические последствия для отдельных компонент или всей системы.

Первый класс задач позволяет описать и анализировать правила функционирования каждой компоненты, а второй класс анализирует взаимовлияние разнородных компонент между собой и системы с внешней средой. Совокупность методов исследования сложных систем и решения задач второго класса и будем относить к системному анализу.

Результатом решения задач первого типа является выбор однородной математической модели (алгебраической, дифференциальной, разностной, логической, реляционной и т. д.), позволяющей наилучшим образом описать процесс преобразования входов в выходы. Для задач второго типа этот метод исследования невозможен из-за разнородности моделей компонент, взаимодействующих между собой и внешней средой по входам и выходам, поэтому возникает проблема объединения разнородных моделей компонент в единую модель системы и системы с внешней средой.

В международных публикациях по созданию моделей процессов, объектов и систем используются два различных термина: modeling – для моделирования однородных процессов, объектов и систем и simulation – для моделирования сложных систем с разнородными компонентами, функционирующими параллельно во времени. Иногда их называют мультиагентными системами. Именно это понятие будет вложено в термин «имитационное моделирование», и именно этому классу систем посвящена данная работа.



■ *Рис. 2. Модель системы, замкнутой на внешнюю среду:*
$X_I=(x_1...x_j...x_J)$, $Y=(y_1...y_i...y_I)$;
$\Theta=(\Theta_1...\Theta_i...\Theta_I)$ – множество разнородных функций преобразования Θ: $X \to Y$

«Вавилонская башня»

В своем стремлении познания законов развития человека, общества и природы человечество накопило множество научных результатов. Они заключаются в выявлении и классификации фактов; описании отдельных явлений, процессов, объектов и систем их классификации; формализации законов функционирования различных процессов, объектов и систем; использовании этих результатов для создания искусственной среды.

Под искусственной средой будем иметь в виду совокупность результатов всех видов деятельности людей: производственную сферу, сферу обслуживания, информационную, энергетическую, коммуникационную среды и т. д. Однако сложность как естественных (человек, общество, природа) так и искусственных систем, которые возникли в процессе развития науки, культуры, образования, техники и технологий, привела к тому, что человеческий интеллект не может во всей полноте охватить эти системы и процессы, проходящие в них. Чтобы глубже проникать в какую-либо конкретную область проблем, человек вынужден все больше абстрагироваться от целостности систем, строя модели для решения отдельных задач, рассматривая единую систему с различных точек зрения. Развитие каждой области науки шло своим путем. Его темп, история и результаты определяются сложностью объекта или системы исследования.

Каждая наука проходит, как известно, необходимые этапы: накопление фактов (знаний), их классификация, построение причинно-следственных связей, описание законов функционирования, строгая (математическая) формулировка законов, позволяющая предсказать результат по исходному состоянию.

Чем более узкая проблема (задача) исследования ставится, чем уже предметная область исследования, тем скорее проходится этот путь. В социально-гуманитарной области объекты исследования многообразны и сложны, а длительности процессов не позволяют провести наблюдения за время жизни наблюдателя. Поэтому результаты исследования различных проблем в разных областях этих наук находятся на очень разных уровнях представления знаний.

Разнообразие форм (языков) представления знаний в различных науках определяется и тем, что для описания явлений или процессов разной природы приходится выбирать те способы представления (описания) законов функционирования систем, которые позволяют наилучшим образом решать поставленную задачу. Социологи пользуются таблицами и графиками при исследовании, например, потребностей населения в чем-то, демографы пользуются диаграммами для выявления распределения каких-либо свойств среди населения, юристы излагают законы в словесных моделях, экономисты – в математических формулах и т. д. Такое же положение в точных науках, где для описания различной природы процессов и объектов используются различные математические аппараты (теории): теоретико-множественные, алгебраические, дифференциальные, вероятностные, логические, реляционные и др.

Выбор типа формализованной модели зависит от типа поставленной задачи и свойств исследуемого объекта. То, что легко решить в одной модели, может оказаться сложно или даже неразрешимо в другой. Простейшим примером этого утверждения может служить модель числового ряда. Ее можно представить в форме римской или арабской (позиционной) системы счисления. Если числовой ряд используется в задаче упорядочения глав в книге, то вполне подойдет и римская модель, но в задаче о стоимости трех килограммов яблок уже никто не станет пользоваться римской системой, так как это отнимет очень много времени (на то, чтобы сначала научиться умножать в этой системе, а потом и подсчитать), в то время как позиционная модель позволяет решить задачу без труда.

Каждая модель адекватна только поставленной задаче. Таким образом, возникновение многообразия моделей при описании сложной системы и ее компонент является объективно необходимым условием процесса познания. Поэтому наши знания о нашем едином мире, его структуре и компонентах, процессах, в нем происходящих, напоминают древнюю притчу о трех слепцах, которых попросили объяснить, что такое слон. Процесс развития Науки напоминает строительство Вавилонской башни, где все строят одну огромную башню – познание единого мира «Жизнь Земли и окружающей ее среды», но перестают понимать друг друга, потому что говорят на разных языках.

На Глобальном Форуме '92 в Рио-де-Жанейро, где руководителями 156 государств обсуждались результаты реализации различных экологических программ, уже было отмечено, что никакие частные экономические, экономико-экологические, эколого-демографические и другие программы не дали ожидаемых результатов и необходимо рассматривать более общие целостные модели Жизни Земли и человеческих сообществ.

В процессе своего развития человечество познавало окружающий мир (естественную среду) и, накапливая информацию (знания), формировало вокруг себя искусственную среду. Информационная среда – накопитель знаний, культуры, общественного сознания – позволяла развиваться далее технологиям, порождавшим все более сложные производственные процессы. Для реализации новых технологий и обеспечения производственных процессов требовалась все более высокая культура: знания, навыки, умение, но и мораль, этика, законы, регулировавшие отношения между людьми. Неравномерность развития различных социумов привела к тому, что в различных регионах мира сложились сообщества с разным типом

культуры, технологий и т. д. Перенесение технологий из одного социума в другой может привести не просто к неудаче, но и к катастрофам, к негативным последствиям для других социумов или даже для всей планеты.

Интересен в этом смысле разговор, имевший место с представителем некоей восточной страны, взявшей одну из информационных технологий в Швеции. На вопрос: «Как работает купленная система?» – он ответил: «Никак!», а на вопрос: «Почему?», последовал ответ: «У нас нет шведов».

Это очень точное определение причины – нет носителей той культуры, для которой была разработана технология. Потребности людей, возможности обеспечить определенные технологии тесно связаны со всей совокупностью сложных общественных структур и процессов, определяющих уровень культуры и законы ее формирования. Обращает внимание тот факт, что крупные японские фирмы тратят до 15 % прибыли на повышение уровня общей культуры и профессионального образования своих работников, чем в большой степени и обеспечивают высокий темп и уровень развития технологий в Японии. Поэтому невозможно рассматривать экономику, технологию и природу в отрыве от человеческой культуры.

Модель любого социума (жители планеты, регионы, поселения, семья, любая человеческая общность на основе производства, соседства, политических и других интересов) должна рассматриваться как сложная единая система Человек–Культура–Технологии–Природа (ЧКТП). Для создания целостной единой модели ЧКТП-системы нет необходимости начинать с нуля. Вся история науки есть история накопления знаний о различных сторонах структур, процессов, свойств как общих для всех ЧКТП-систем, так и определяющих специфику этнических, политических, экономических, экологических, культурных и других условий конкретных социумов, сообществ, народов, регионов, государств.

Для создания единой модели ЧКТП-системы необходимо найти средства объединения уже существующих разнородных по языкам описания и уровню формализации частных моделей подсистем и компонент в единую систему и определить недостающие звенья в этой модели.

Особенно важно, чтобы оказались востребованными результаты, накопленные в социально-гуманитарных науках, которые изучали закономерности развития культуры, процессы возникновения и законы формирования потребностей человека как основы развития любой человеческой цивилизации. Однако результаты, выраженные словами, несут большую неоднозначность и неопределенность интерпретации, что отличает социально-гуманитарное знание от знания, сформированного точными науками, и порождает проблему разработки способов и средств объединения результатов (моделей) точных и гуманитарных наук.

Структура гибких технологических систем

Чтобы каждая частная модель подсистемы, компоненты, проблемы или задачи могла найти четкое место в единой модели, нужно построить структуру целостной системы. Для этого определим свойства класса рассматриваемых систем. Характерными свойствами таких систем являются:

одновременное существование множества различных целей, ограничений и критериев функционирования;

наличие схем поведения (технологических процессов) для достижения каждой цели при заданных ограничениях и критериях функционирования;

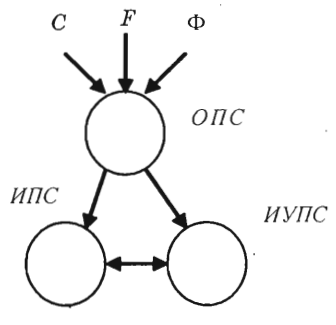
способность функционировать в условиях изменения целей, ограничений и критериев функционирования (изменять схему поведения).

Назовем такой класс систем гибкими технологическими системами (ГТС). Под гибкостью будем понимать способность системы подстраиваться под изменение целей, ограничений и критериев при сохранении жизнеспособности системы. Под технологическими процессами будем подразумевать способы и последовательность действий для достижения целей.

Любую многоцелевую систему можно представить как совокупность трех функциональных взаимосвязанных подсистем – организационной (ОПС), информационно-управляющей (ИУПС) и исполнительной (ИПС) (рис. 3).

Организационная подсистема – это совокупность средств и методов, определяющая выбор целей, ограничений и критериев функционирования (поведения) системы на основе сформированной цели существования (жизни) системы.

Понятие цели употребляется в двух различных смыслах. Первый – цель существования (жизни) системы, которая закладывается при ее возникновении (создании) и существует в течение всей жизни системы. Например, это поддержание жизни



■ ***Рис. 3.*** *Функциональная структура многоцелевых систем:*
C – множество целей;
F – множество критериев оценки достижения целей;
Ф – множество ограничений на способы достижения целей

рода, вида и индивидуума в объектах живой природы; выпуск различных автомашин на автозаводе; формирование специалистов определенных профессий и квалификации в вузах; обслуживание самолетов, пассажиров и грузов в аэропорту.

Система существует, пока она поддерживает заданные цели существования, или жизни. При изменении всех целей существования будем считать ее умершей. Она может жить уже в другой ипостаси, например, как умерший человек в соответствии с одной из индийских религий.

Второе понятие – цель поведения, или функционирования системы. Она определяется на конечном интервале времени в зависимости от внутреннего состояния системы и ситуации в окружающей среде. При этом цель может быть обобщенной, удаленной, к которой система может стремиться в течение некоторого времени – асимптотическая цель или «идеал», и конкретная (C), четко достижимая, выраженная в конкретных характеристиках (атрибутах) в терминах реляционной алгебры: какие объекты (d) подлежат изменению, в каком количестве (ε), каким способом (θ) и за какой период ($\tau_{нач}$, $\tau_{кон}$):

$$C = \langle d, \varepsilon, \theta, \tau_{нач}, \tau_{кон} \rangle.$$

Для достижения цели существования системы или поддержания поведения, обеспечивающего достижение этой цели, организационная подсистема должна определять стратегию поведения системы, ставить последовательные цели ее поведения с учетом изменения ситуаций в окружающей среде и внутреннем состоянии системы.

Таким образом, функцией оргсистемы является стратегическое обеспечение цели существования, выработка «идеала» на основе анализа состояния системы и внешней среды и определения цепочки целей поведения, ведущих систему к «идеалу». Кроме того, ОПС обеспечивает принятие решений в экстремальных ситуациях для поддержания или сохранения жизнеспособности системы. ОПС задает «правила игры» на некотором интервале времени для ИПС и ИУПС.

Традиционно в теории управления используют структуру из двух подсистем: объект управления и система управления. Это связано с тем, что данная теория рассматривала технические системы, если и сложные, то все-таки с определенным набором целей. В этом случае, цели существования и цели поведения совпадают и закладываются жестко в структуру исполнительной (объект управления) и информационно-управляющей (система управления) подсистем. Например, питьевой автомат. Цель существования – выдавать по запросу (монета) нужную порцию воды и сиропа. Цель заложена в конструкцию исполнительной и управляющей частей автомата.

Самонаводящиеся ракеты тоже относятся к этому классу. У такой системы изменение цели не является признаком принадлежности к классу ГТС, так как под целью C здесь понимается достижение целевого объекта d, движущегося в пространстве, а поведение ракеты θ – достижение координаты целевого объекта и, возможно, его уничтожение. Изменится не цель, а атрибут объекта.

Информационно-управляющая подсистема – совокупность средств и методов, обеспечивающих взаимодействие системы с внешней средой и компонентов системы между собой в процессе достижения цели поведения, заданной ОПС в соответствии с текущей ситуацией и критериями функционирования.

Информационно-управляющая подсистема обеспечивает наблюдаемость изменений состояний (событий) в любом компоненте системы, осуществляет контроль соответствия фактических состояний целевым (заданных ОПС) на основе результатов контроля, принимает решение о требуемых изменениях в правилах функционирования компонентов системы.

ИУПС обеспечивает функционирование ИПС для достижения целей, поставленных ОПС, в соответствии с правилами (ограничениями и критериями), установленными ОПС.

Исполнительная подсистема – совокупность всех исполнительных средств, способная обеспечить выполнение всех технологических процессов, необходимых для достижения целей поведения (функционирования) системы.

Наличие всех трех подсистем характерно для гибких систем, т. е. систем, способных функционировать в условиях изменения целей, ситуаций (ограничений) и критериев поведения.

Рассмотрим в качестве примера Университет. Цель его жизнедеятельности– готовить образованных специалистов в определенных областях знаний, культурных людей и законопослушных граждан государства. Цели функционирования Университета – сколько, каких специалистов, в какие сроки, при каких ограничениях выпускать и какими критериями оценивать результат деятельности.

Ректор и Совет Университета на основе цели жизнедеятельности, заложенной при учреждении Университета и социального заказа общества, определяют цели функционирования, т. е. являются организационной подсистемой.

Исполнительной подсистемой являются весь профессорско-преподавательский и учебно-вспомогательный состав, материально-хозяйственная и учебная часть. Информационно-управляющей подсистемой являются ректорат, деканаты, заведующие кафедрами, управленцы административно-хозяйственного и учебного аппарата, общественные, научные и прочие структуры, обеспечивающие взаимодействие всех участников процесса обучения в соответствии с поставленными ОПС целями жизнедеятельности Университета в процессе продвижения к заданному на данный период «идеалу» – поддержание работоспособности

ИПС в соответствии с заданными учебными планами и расписаниями.

Такая структуризация позволяет определить необходимые функции, общие в любых системах. При построении иерархических систем обычно роль ОПС выполняет ИУПС более высокого уровня, ибо именно она обычно задает «правила игры» (цели, ограничения, критерии) для ИУПС и ИПС нижнего уровня. Например, любой факультет Университета можно рассматривать как ГТС, для которой роль ОПС играет ректорат, информационно-управляющей системой является деканат, а исполнительной системой – кафедры факультета. Для кафедры функции ОПС выполняет деканат, ИУПС – заведующий кафедрой, ИПС являются весь преподавательский и учебно-вспомогательный состав кафедры.

Каким бы образом ни структурировать систему, каждая подсистема и компонента на любом уровне иерархии будут, в свою очередь, состоять из таких же трех функциональных подсистем до тех пор, пока вариативность (изменяемость) целей, ограничений и критериев остается их свойством.

Для чего нужны модели?

Человеческий интеллект... Возможность ставить и достигать цели при изменяющихся обстоятельствах, способность выбирать из множества целей те, которые скорее ведут к желаемому состоянию, адаптация к изменениям в среде и внутреннем состоянии путем изменения среды или собственного состояния. Казалось бы, неограниченные возможности. И человек всегда ставил цели и принимал решения, как построить свое поведение или изменить среду и объекты, чтобы достичь желаемой цели. Но мир вокруг него усложнялся, естественная и искусственная среда становились все более многообразными, процессы и явления все более тесно связывались между собой, и принимать решения в новых условиях становилось все сложнее, а главное, опаснее. Если человек принимал решения в малой общности, в малом социуме, при простой технологии его неверные решения могли, в крайнем случае, уничтожить этот социум. Но когда человек принимает решение в мире, где все социумы тесно связаны едиными источниками дефицитных ресурсов, а технологии таковы, что способны уничтожить весь мир, принимать решения требуется с большей осторожностью, не забывая о связности системы, о том, что изменение в одной части системы может вызвать последствия в других.

Человеческий интеллект, его уникальные способности обеспечиваются следующими функциями:

получение информации о внешней среде и внутреннем состоянии самого субъекта;

формирование модели внешней среды и внутреннего состояния субъекта на основе полученной информации;

целеполагание – формулирование «идеала» – глобальной цели жизнедеятельности (чаще неформальное и нечеткое), последовательности целей поведения, ведущей к глобальной цели системы, и критериев оценки ее достижения при сохранении жизнеспособности в соответствии с имеющейся моделью внешней среды и внутреннего состояния субъекта;

поиск решения – функция, обеспечивающая формулировку проблемы (задачи, что именно требуется разрешить), определение текущей ситуации в соответствии с построенной моделью, выбор методов разрешения конфликтных ситуаций из уже имеющихся и/или формулировка новых при возникновении новой конфликтной ситуации, накопление методов решения;

реакция на ситуацию – изменение внешней среды или внутреннего состояния для разрешения конфликтной ситуации на основе принятого решения, что требует наличия исполнительной системы;

накопление опыта – построение и запоминание системы связей между целью, ситуацией, решением, результатами и оценкой результата;

обучение – упорядочение опыта, выделение наиболее удачных сочетаний цель – ситуация – решение – реакция – оценка результата.

Эти функции в той или иной мере существуют у любой живой системы. Человеческий интеллект отличается возможностью выполнения всех перечисленных функций над абстрактными объектами (образами, моделями), сформировавшимися на основе накопленного опыта и обучения.

Целеполагание относится к функции ОПС, формирование модели, накопление опыта, обучение и поиск решения выполняются ИУПС, а получение информации и реакция на ситуацию – функция ИПС.

Все эти функции необходимы и тогда, когда решение принимает не один субъект, а коллектив или целый социум. Поэтому, когда возникла ситуация, для разрешения которой человеческого интеллекта стало недостаточно для сохранения жизнеспособности, человечество стало интенсивно искать способы усиления своего интеллекта, пытаясь создать коллективный интеллект. Все способы обмена информацией между людьми в процессе их жизнедеятельности были направлены на создание коллективного интеллекта, чтобы расширить возможность и ускорить выполнение всех перечисленных функций [1].

При появлении вычислительных машин стало казаться, что искусственный интеллект будет скоро создан, и были разработаны «универсальные решатели», «универсальные доказатели» теорем. Но затем пришло понимание, что создать систему, обладающую всеми перечисленными свойствами в рамках однородной модели (на основе одного математического аппарата или теории) невозможно. Все модели в области искусственного интеллекта были лишь усилителями отдельных функций ин-

теллекта человека, подобно микроскопу, телескопу, очкам, которые являются усилителями зрения, а не искусственным зрением.

Поэтому при решении сложных задач следует позаботиться об усилении тех интеллектуальных функций, которые необходимы в каждом конкретном случае. Чаще всего возникают проблемы построения модели рассматриваемой системы и окружающего мира, адекватной возникшей проблеме. Поэтому большинство исследовательских проблем связано именно с построением таких моделей. Но так как интеллект человека количественно ограничен и не может одновременно удерживать в активной памяти множество параллельно существующих и функционирующих компонент системы, человек всегда вычленяет в сложной системе или процессе ту его часть, которая в наибольшей степени (с его точки зрения) влияет на решение поставленной проблемы или разрешения конфликта. Такое вычленение редко возможно обосновать, так как в многосвязных многокомпонентных системах отдельные слабые связи могут, в конечном счете, вызывать в системе очень большие последствия.

Построение моделей сложных систем и процессов требует совместного участия исследователей, экспертов из разных наук и научных направлений для описания способов функционирования компонент системы, различных по своей природе, структуре и способам функционирования, а каждая наука имеет свой язык (аппарат, способы исследования и выражения результатов). Это порождает необходимость сопряжения разнородных моделей отдельных подсистем и процессов, в них протекающих, выраженных на разных языках, в единую модель.

Особенно важно создание целостных моделей для лиц, принимающих решение (ЛПР) в больших ЧКТП-системах: крупных фирмах, производственных объединениях, городах, регионах, государствах и других системах.

Для принятия какого-либо решения, обеспечивающего очередную цель поведения управляемой системы, ведущую к глобальной цели, ЛПР должно сформировать модель системы, ее возможные состояния, способы ее возможного и желаемого функционирования:

получить информацию о состоянии той среды, в которой «живет» система, и о ее собственном внутреннем состоянии на текущий момент относительно всех атрибутов, существенных для поставленной задачи;

построить статическую и динамическую модели среды и системы;

построить способы возможного функционирования, т. е. все множество допустимых состояний: ситуация – цель – решение – результат – оценка;

построить способ желаемого функционирования (обучить жить по лучшим законам), выбрать из множества способов желаемые;

сформировать способы выбора очередной цели поведения, ведущей к «идеалу» или обеспечивающей движение в его направлении на основе анализа состояний внешней среды или внутреннего состояния;

разработать способы принятия решений, разрешения конфликтов (определить новые способы, правила, законы функционирования компонент исполнительной системы и/или способы взаимодействия между ними);

реализовать принятое решение в действиях исполнительной системы, т. е. заставить компоненты исполнительной системы функционировать в соответствии с новыми правилами.

Таким образом, для поддержки принятия решения при управлении сложными системами необходимо иметь инструментальную систему усиления интеллектуальных способностей ЛПР. Такой инструментальной системой поддержки принятия решений может служить компьютерная имитационная модель, позволяющая ЛПР проверить свои решения и их последствия для всей системы в целом, определить причины и компоненты, не удовлетворяющие критериям достижения целей целостной системы без экспериментирования на людях и природе.

Структура модели ЧКТП-системы

При рассмотрении любой системы, для которой требуется построить компьютерную имитационную модель, необходимо определить цели ее существования и цели поведения, обеспечивающие достижение цели существования. Возьмем для примера создание любой фирмы. Предположим, при ее создании была поставлена цель C_0 – получение прибыли в размере ε_0 с помощью техпроцесса θ_0 за период времени с $\tau_{н0}$ по $\tau_{к0}$. Представим цель в реляционной форме:

$$C_0 = \langle d_0, \varepsilon_0, \theta_0, \tau_{н0}, \tau_{к0} \rangle,$$

где C_0 – цель (получение прибыли); d_0 – целевой объект (прибыль); ε_0 – количество прибыли, которое хотим получить; θ_0 – способ получения прибыли; $\tau_{н0}$ – начальный момент периода получения прибыли; $\tau_{к0}$ – конечный момент периода получения прибыли.

В сформулированной цели не определен способ получения цели θ_0. Следовательно, возникает следующая цель C_1 – получить информацию о том, каким способом можно обеспечить требуемую прибыль. Для этого нужно провести маркетинг, на основании результата которого выберем θ_0 – способ получения прибыли.

Сформулируем следующую цель:

$$C_1 = \langle d_1, \varepsilon_1, \theta_1, \tau_{н1}, \tau_{к1} \rangle,$$

где C_1 – цель (получение маркетинговой информации); d_1 – целевой объект (маркетинговая информация); ε_1 – количество маркетинговой информа-

ции, которую необходимо обработать; θ_1 – способ получения маркетинговой информации (маркетинг); $\tau_{н1}, \tau_{к1}$ – периода обработки маркетинговой информации.

Для проведения маркетинга требуется специалист – маркетолог. Следующая цель C_2 – найти маркетолога:

$$C_2=\langle d_2, \varepsilon_2, \theta_0, \tau_{н2}, \tau_{к2}\rangle,$$

где C_2 – цель (найти маркетолога); d_2 – целевой объект (маркетолог); ε_2 – количество маркетологов (хотя бы одного); θ_2 – способ поиска специалистов; $\tau_{н2}$ – начальный момент периода; $\tau_{к2}$ – конечный момент периода поиска специалистов.

Для найма маркетолога нужны финансы. И возникает цель C_3:

$$C_3=\langle d_3, \varepsilon_3, \theta_3, \tau_{н3}, \tau_{к3}\rangle,$$

где C_3 – цель (найти финансы); d_3 – целевой объект (финансы); ε_3 – количество финансов; θ_3 – способ поиска, получения финансов; $\tau_{н3}$ – начальный момент периода; $\tau_{к3}$ – конечный момент периода, отведенного на поиск.

Для поиска финансиста используем цель C_2, где определен θ_2 – способ поиска специалистов. Теперь в результате достижения всех поставленных целей будет известен способ θ_4, которым можно получить прибыль, определенную в исходной цели C_0:

$$C_4=\langle d_4, \varepsilon_4, \theta_4, \tau_{н4}, \tau_{к4}\rangle,$$

где C_4 – цель (произвести тот продукт, который и принесет прибыль); d_4 – целевой объект, который и будет выходным объектом (потоком) проектируемой фирмы, за который фирма сможет получить прибыль (целевой объект может выражаться материальным, информационным, энергетическим и другими потоками); ε_4 – величина (количество) целевого потока, необходимого для получения заданной прибыли; θ_4 – способ производства целевого потока; $\tau_{н4}, \tau_{к4}$ – начальный и конечный моменты периода производства и сбыта продукта.

Для производства продукта необходимо иметь проект производственной системы, который определит структуру и состав исполнительной системы фирмы для реализации технологического процесса θ_4, т. е. возникает цель C_5:

$$C_5=\langle d_5, \varepsilon_5, \theta_5, \tau_{н5}, \tau_{к5}\rangle,$$

где C_5 – цель (разработать проект производственной системы); d_5 – целевой объект (проект производственной системы); ε_5 – количество (хотя бы один проект); θ_5 – способ проектирования производственной системы; $\tau_{н5}, \tau_{к5}$ – период проектирования.

Для реализации проекта необходимо достигнуть цели, обеспечивающие возможность функционирования производственной системы:

$$C_6=\langle d_6, \varepsilon_6, \theta_6, \tau_{н6}, \tau_{к6}\rangle,$$

где C_6 – цель (обеспечение места размещения производственной системы в пространстве и создание ее геоинформационной системы GIS); d_6 – целевой объект (участок земли, здания, помещения и их карта, планы); ε_6 – количество, необходимое для размещения производственной системы; θ_6 – способ поиска, покупки или аренды, заключения договоров, составление карт, планов; $\tau_{к6} - \tau_{н6}$ период обеспечения размещения производственной системы в пространстве и создание ее GIS;

Теперь имеем пространство для размещения исполнительной системы фирмы. Возникает следующая цель: обеспечить разрабатываемую фирму специалистами и исполнительными средствами в соответствии с проектом d_6. Для обеспечения специалистами уже определена цель C_2. Сформулируем следующую цель C_7 – организовать функцию снабжения производственной системы всеми исполнительными средствами и расходуемыми ресурсами:

$$C_7=\langle d_7, \varepsilon_7, \theta_7, \tau_{н7}, \tau_{к7}\rangle,$$

где C_7 – цель (обеспечение производственной системы исполнительными средствами); d_7 – целевой объект (требующиеся исполнительные средства); ε_7 – количество средств, необходимое для обеспечения функционирования производственной системы; θ_7 – способ поиска, заказа, покупки, доставки и установки исполнительных средств; $\tau_{н6}, \tau_{к7}$ – период обеспечения производственной системы всеми исполнительными средствами.

Для безопасного функционирования системы в рамках законности необходимо юридическое обеспечение системы:

$$C_8=\langle d_8, \varepsilon_8, \theta_8, \tau_{н8}, \tau_{к8}\rangle,$$

где C_8 – цель (юридическое обеспечение производственной системы); d_8 – целевой объект (юридическая информация); ε_8 – количество необходимой юридической информации для функционирования производственной системы; θ_8 – способ поиска и получения юридической информации; на $\tau_{н8}$ и $\tau_{к8}$ – период обеспечения юридической информацией производственной системы.

Следующее условие функционирования системы – безопасность. Цель C_9 – обеспечить безопасность как системы от внешнего мира (окружающей среды), так и внешнего мира от вредного влияния проектируемой производственной системы.

$$C_9=\langle d_9, \varepsilon_9, \theta_9, \tau_{н9}, \tau_{к8}\rangle,$$

где C_9 – цель (обеспечение безопасности производственной системы); d_9 – целевой объект (степень необходимой безопасности); ε_9 – количественная характеристика безопасности, необходимой для функционирования производственной системы; θ_9 – способ обеспечения безопасности по различным аспектам и параметрам; $\tau_{н9}$ и $\tau_{к9}$ – начальный и конечный моменты периода обеспечения безопасности.

При достижении всей последовательности перечисленных целей создаваемая система должна быть способна достигать цели ее жизнедеятельности, в данном случае, достигать прибыли в заданном количестве за заданный период времени. Но для этого нельзя забыть еще одну цель системы: сообщить миру о своем существовании и выпускаемом продукте:

$$C_{10}=\langle d_{10}, \varepsilon_{10}, \theta_{19}, \tau_{н10}, \tau_{к10}\rangle,$$

где C_{10} – цель (обеспечение рекламы фирмы и производимого ею продукта); d_{10} – целевой объект (рекламная информация); ε_{10} – количество необходимой рекламы; θ_{10} – способ обеспечения рекламы по различным аспектам и параметрам продукта и системы; $\tau_{к10}$–$\tau_{к10}$ –период обеспечения рекламой.

Так как в рассматриваемом примере не учитывались никакие специфические свойства системы, эту последовательность целей при создании любой системы можно принять за минимально необходимый инвариант набора целей на верхнем уровне описания модели ГДТС или ЧКТП-систем.

Отметим, что все десять целей были сформулированы как последовательность целей, ведущая к созданию фирмы. Это цели поведения создателей фирмы, в которую заложена цель жизнедеятельности $C_0=\langle d_0,\varepsilon_0,\theta_0,\tau_{н0},\tau_{к0}\rangle$. При функционировании созданной фирмы эта же последовательность целей будет обеспечивать поддержание цели жизнедеятельности фирмы C_0 на протяжении всего жизненного цикла фирмы. Действительно, при функционировании любой фирмы необходимо вести постоянный маркетинг, заниматься подбором кадров, вести финансовые операции, выполнять основной технологический процесс выпуска целевой продукции, поддерживать аренду земли и недвижимости, обеспечивать снабжение, безопасность фирмы, юридические услуги, а также заниматься рекламой.

Любой структурный компонент системы, предназначенный для достижения своей цели, можно характеризовать множеством целей C, входными и целевыми потоками (d_0, $d_ц$) и правилами преобразования (техпроцессами θ) входных потоков в целевые. Следовательно, каждой цели должен быть поставлен в соответствие компонент в структуре системы. Инвариантное представление любой подсистемы или компоненты приведено на рис. 4.

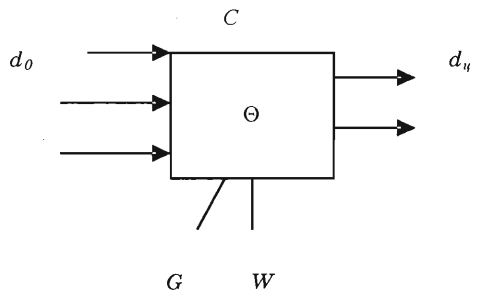
Введение инвариантного представления любой подсистемы или компоненты позволяет построить инвариантное математическое описание всех ее характеристик. Следовательно, каждой из десяти целей инвариантного набора должен соответствовать свой структурный компонент в создаваемой модели любой системы. Так как компонент выполняет способ преобразования целевого потока, т. е. обеспечивает способность выполнения всех технологических процессов, необходимых для достижения целей поведения (функционирования), то этот набор из десяти компонент и составляет инвариант функциональной структуры исполнительной подсистемы (ИПС) гибких дискретных технологических систем.

Этот инвариант функциональной структуры исполнительной подсистемы ГДТС и примем за структуру модели ИПС любой системы класса ГДТС.

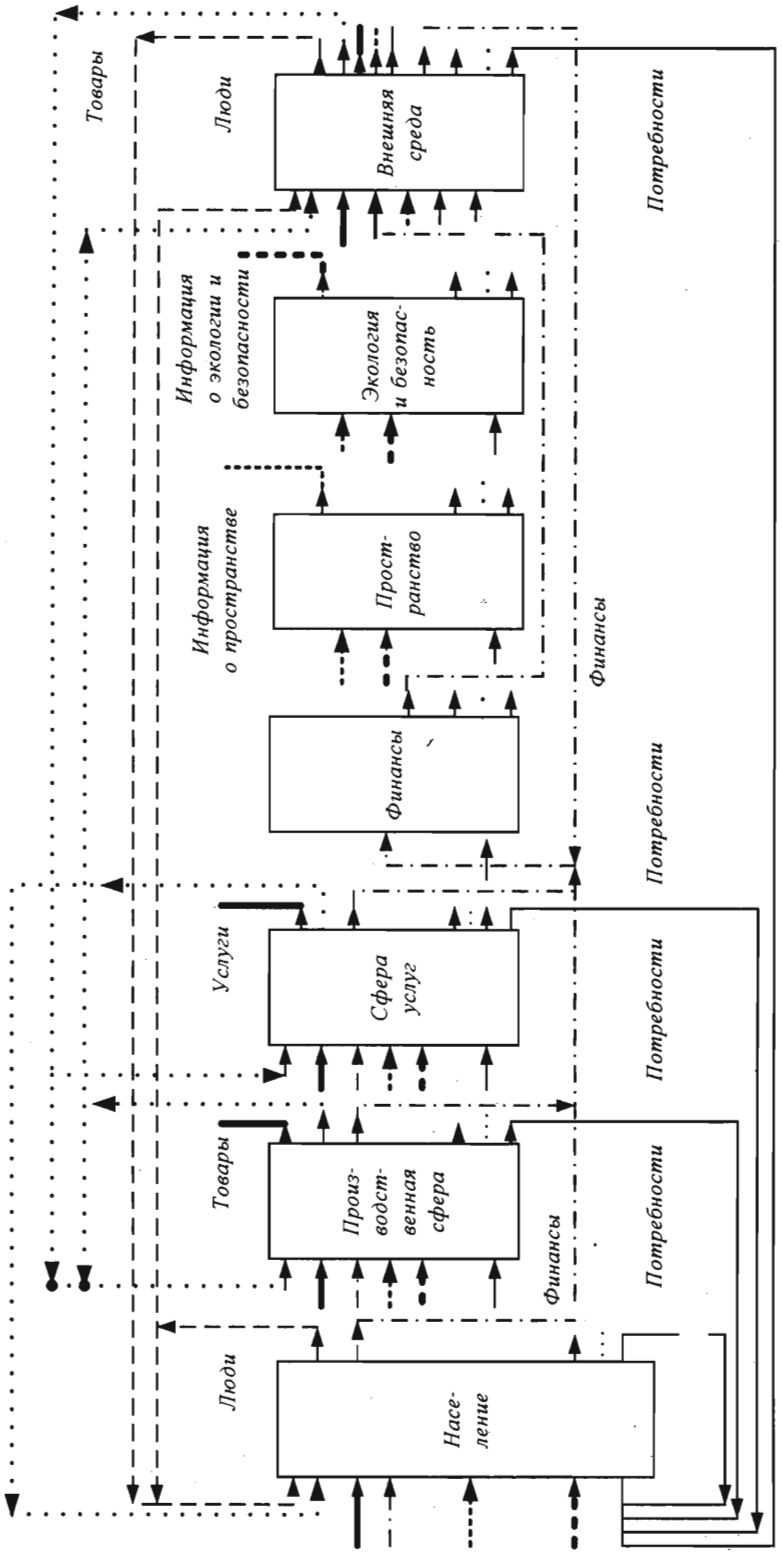
Однако так как эта модель предназначена для создания систем поддержки принятия решений или усиления интеллектуальных способностей ЛПР при принятии управленческих решений в сложных многоцелевых системах с разнородными компонентами, то приходится учитывать, что число одновременно активно воспринимаемых и обрабатываемых человеком объектов не должно превышать девяти. Поэтому, чтобы человеку было удобно воспринимать всю модель как единое целое, переструктурируем инвариантный набор компонент таким образом, чтобы инвариантная часть содержала на верхнем уровне иерархической структуры модели ИПС только семь компонент (рис. 5).

Так как при моделировании ГДТС необходимо учитывать влияние внешней среды на систему и системы на внешнюю среду, то модель внешней среды должна входить в моделирующую систему как равноправная компонента ИПС моделирующей системы. Включим блок «Внешняя среда» в структуру моделирующей системы на верхнем уровне иерархии. Выделим сначала те целевые компоненты, у которых потоки имеют специфические свойства, общие для широкого класса систем.

Начнем с цели C_2 – поиск маркетолога, потом финансиста, потом специалистов, входящих в исполнительные средства производственной системы и системы управления. Этот компонент ИПС должен отражать информацию о человеке и процессах его жизнедеятельности в системе – и как биологического и социального объекта, и как спе-



■ ***Рис. 4.*** *Инвариантное представление любой подсистемы или компоненты:*
C – множество целей компоненты; $d_ц$ – целевой поток, обрабатываемый компонентой; Θ – техпроцесс или способ изменения свойств целевого потока от начального состояния d_0 до целевого состояния $d_ц$;
G, W – исполнительные и вспомогательные средства, необходимые для выполнения техпроцесса и достижения целей компоненты



■ *Рис. 5. Структура Исполнительной подсистемы ГДТС:*
– – – – – люди; –·–·–·– финансы;
——— услуги; ——— потребности;
. . . . . товары; ---------- информация о пространстве;
■ ■ ■ ■ информация о экологии и безопасности

циалиста, т. е. исполнительного средства в ИПС. В любой системе ЧКТП этот компонент всегда будет присутствовать и обладать общими функциями. Выделим его как блок «Люди» в структуре модели.

Везде будет присутствовать основная цель C_4 обеспечения производственного процесса, выпускающего целевой поток системы, предназначенный для внешней среды. Этот компонент ИПС для различных систем будет иметь наибольшую специфику описания функционирования, так как связан с различными технологиями. Введем его в инвариант как «Производственная сфера».

В большей степени общими свойствами и функциями для всех систем обладает C_3 – обработка финансовых потоков. Так как финансовый поток – это чисто информационный поток со специфической обработкой на основе общих для многих систем правил и законов, то его тоже целесообразно оставить на верхнем уровне иерархии ИПС. Назовем его «Финансы».

Обработка информационных потоков о территориальном, пространственном размещении системы, ее подсистем и компонент (C_6) обладает большой общностью структур и функций для различных систем. Используем для этого структурного блока принятую аббревиатуру GIS – геоинформационная система.

Обеспечение безопасности производственной системы и внешней среды (C_9) как блок, наиболее связанный с внешней средой, тоже имеет смысл оставить на верхнем уровне. Назовем этот блок «Безопасность». Под безопасностью понимаются способы наблюдения и устранения всех видов опасности жизнедеятельности как со стороны моделируемой системы находящимся в ней людям и внешней среде, так и со стороны внешней среды моделируемой системе, ее подсистемам и компонентам.

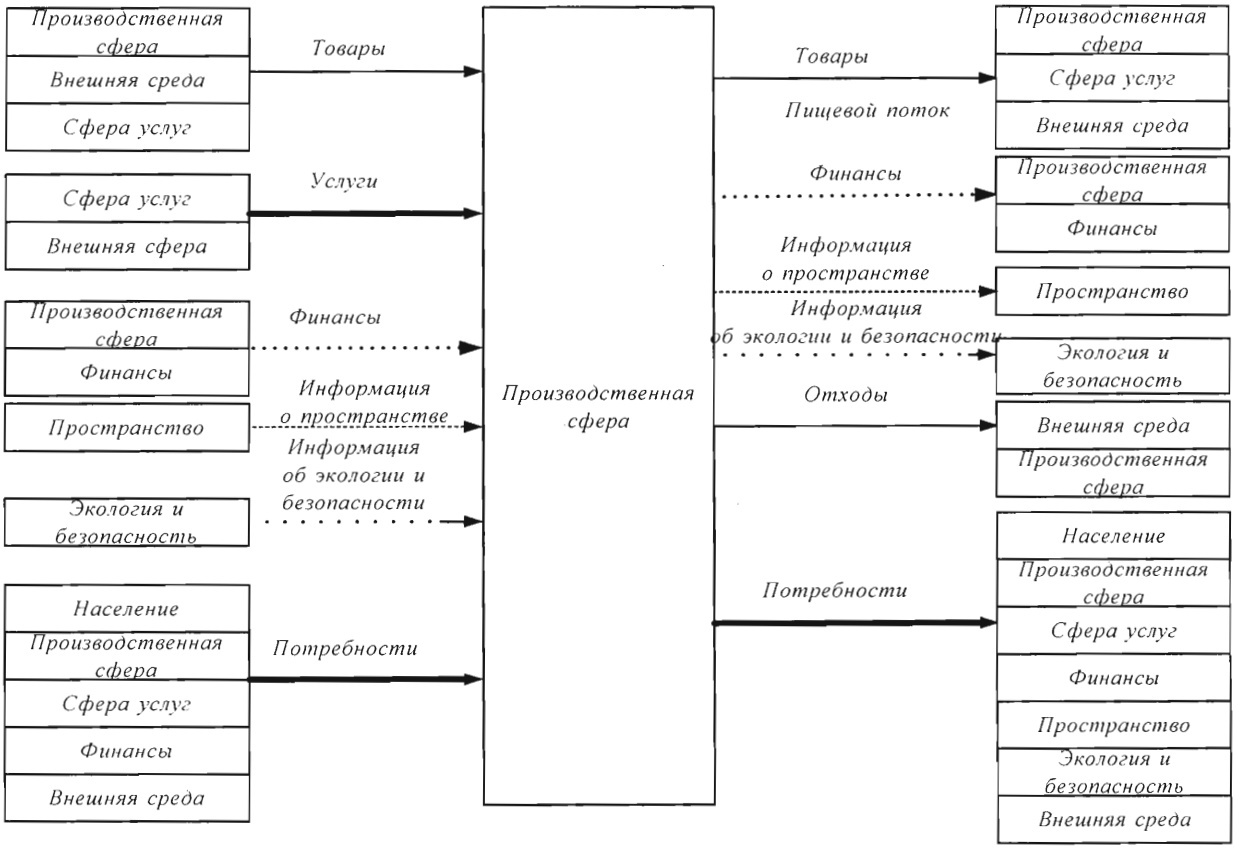
Введем на верхний уровень компонент «Сфера услуг» и объединим в него на следующем уровне оставшиеся целевые блоки, обеспечивающие услуги, требуемые для функционирования остальных подсистем: информационное обслуживание (C_8 – юридические услуги, C_1 – маркетинговые услуги и C_{10} – рекламные услуги), C_7 – услуги материального снабжения, C_5 – проектные разработки. На более низком уровне иерархии в эту же подсистему войдут услуги энергоснабжения, обеспечения здоровья, обучения, коммуникационные и другие услуги.

Получившаяся структура исполнительной подсистемы, состоящая из семи компонент, является инвариантом для всего класса ЧКТП-систем на верхнем уровне иерархической структуры модели ИПС (см. рис. 5). Так как эта структура описывает только самые общие свойства любых систем ЧКТП, то назовем эти компоненты подсистемами ИПС ЧКТП-систем. На следующем уровне иерархической структуры компоненты подсистем будут обладать большим разнообразием целей, целевых потоков и технологических процессов. На верхнем уровне самым важным свойством модели является замкнутость системы по всем целевым потокам между всеми подсистемами моделируемой системы и внешней средой. Поэтому первой задачей при построении модели любой ГДТС должна быть проверка ее замкнутости по всем потокам, так как каждый входной поток в подсистему должен быть выходным из какой-то подсистемы или внешней среды, а каждый выходной должен служить входом какой-то подсистемы или внешней среды, т. е. ни один поток не может поступить «ниоткуда» и уходить «никуда».

Построение структуры потоков любой подсистемы начинается с определения ее основной цели и целевого потока. Для получения целевого потока подсистеме необходимо получить от других подсистем или внешней среды множество исходных ресурсов, информационных и энергетических потоков и услуг, из которых подсистема и выработает по ее техпроцессу основной выходной поток. Эту информацию и выдает каждая подсистема в виде потока потребностей, адресованных другим подсистемам. В свою очередь, подсистема получит от других подсистем и требуемые потоки на свой вход и потребности других блоков, в ответ на которые она выдаст дополнительные выходные потоки.

На рис. 6 показана структура потоков одной из подсистем «Производственная сфера»: потоки, которые необходимы на ее входе для выработки целевых потоков, и их источники (слева), целевые потоки на выходе подсистемы «Производственная сфера» и их подсистемы-приемники (справа). Имея такое описание любой подсистемы, легко осуществить проверку замкнутости потоков в системе. Каждая подсистема должна описываться независимо, так как для описания разных подсистем требуются эксперты из разных отраслей знания, и проверка на замкнутость позволяет установить, все ли эксперты правильно (одинаково) представляют себе взаимодействие своей подсистемы с другими подсистемами и внешней средой.

Полная замкнутость модели ИПС делает наглядным ее свойство самоуправления. Каждая подсистема для обеспечения функционирования в процессе достижения своей цели вырабатывает информационный поток о потребностях, адресуя свои потребности соответствующим подсистемам. Так как каждая подсистема тоже состоит из трех подсистем (ОПС, ИУПС, ИПС) и, следовательно, имеет свою организационную подсистему, которая воспринимает потребности других подсистем как поток целей на своем входе и изменяет, соответственно, правила функционирования своей ИУПС для формирования потоков, удовлетворяющих эти системные потребности. Если правила функционирования заложены во все подсистемы правильно относительно внешней ситуации и набора целей, то при малых отклонениях внешней ситуации и набора целей,



■ ***Рис. 6.*** *Структура потоков подсистемы «Производственная сфера»*

система может устойчиво работать без внешнего управления. (При краткосрочном отсутствии руководства фирма работает за счет самоуправления, поддерживая целевые потоки путем обмена потоками потребностей).

Для инвариантного описания всех подсистем различными специалистами разработаны математические основы общей теории гибких дискретных технологических систем (ГДТС) на основе реляционной алгебры (алгебры отношений или просто таблиц), так как табличное представление любых зависимостей наиболее универсально для представителей всех наук. Математические основы общей теории содержат описание статической и динамической реляционной модели компонент, метод декомпозиции компонент в зависимости от решаемой задачи, функций связи в иерархической структуре подсистем (вертикальных связей), функции связей между подсистемами в инвариантной структуре ГДТС (горизонтальные связи). Математические основы общей теории могут служить хорошей базой создания программных инструментальных средств для построения имитационных моделей сложных систем типа ЧКТП как единой системы, все компоненты которой строятся на единых принципах.

Литература

1. **Лотман Ю. М.** Семиосферы. Культура и информация. – СПб.: «Искусство-СПб», 2001. – 704 с.
2. **Перовская Е. И.** Основные принципы построения моделей сложных гибких дискретных технологических систем // Системный анализ в проектировании и управлении: Труды VIII международн. научно-практич. конф. – СПб.: Изд-во СПбГТУ, 2004. – С. 126–130.
3. **Перовская Е. И.** Создание глобальных моделей социумов для проверки управленческих решений и их последствий // Системный анализ в проектировании и управлении: Труды VII международн. научно-практич. конф. – СПб.: Изд-во СПбГТУ, 2003. – С. 186–190.
4. **Perovskaya E. I.** Support of acceptance of the administrative decisions in systems Humanity–Culture–Technology–Nature / Instrumentation in ecology and human safety 2002. – St.Petersburg, 2002. – С. 194–196.
5. **Перовская Е. И., Фетисов В.** Автоматизация гибких дискретных систем. – Л.: Изд-во ЛГУ, 1986. –183 с.
6. **Перовская Е. И.** Имитационные модели для поддержки принятия решений в системах Человек–Культура–Технологии–Природа // Системный анализ в проектировании и управлении: Труды V международн. научно-практич. конф. – СПб.: Изд-во СПбГТУ, 2001. – С. 177–181.

УДК 681.3

РЕАЛИЗАЦИЯ БИБЛИОТЕКИ ИМИТАЦИОННЫХ МОДЕЛЕЙ КАК НАБОРА ОБОБЩЕННЫХ КОМПОНЕНТ

А. В. Колотаев,
ведущий программист
ОАО «Транзас технологии»

В статье рассматривается применимость методов обобщенного программирования для реализации библиотеки имитационных моделей на языке C++. Описываемый подход представляется перспективным для разработки моделей больших и сложных систем, поскольку позволяет сочетать эффективность имитационных программ с широкими возможностями повторного использования модулей.

Applicability of generic programming techniques to develop simulation models C++ library is discussed. The approach described seems to be valuable for large and complex systems development since it allows to create highly reusable modules without sacrifying their efficiency.

Введение

Представление имитационной модели в виде набора взаимодействующих компонент является общепринятой практикой и имеет большую историю (например, один из первых языков объектно-ориентированного программирования Simula-67 предоставлял средства для модульной декомпозиции моделей еще в конце 60-х годов прошлого века). Преимущества модульного подхода хорошо известны. Во-первых, это возможность отразить в коде имитационной программы структуру моделируемой системы, что, несомненно, способствует пониманию программы и облегчает таким образом ее сопровождение. Во-вторых, такой подход открывает возможности для использования одного модуля в качестве составного элемента для построения разных имитационных моделей, т. е. для повторного использования модулей; это снижает затраты на построение и отладку новых имитационных моделей. Чем сложнее конструируемая модель, тем ярче проявляются преимущества модульного подхода.

Одним из наиболее распространенных языков для создания имитационных программ сложных и больших систем (например, телекоммуникационных сетей – ТКС) является язык C++, что обусловлено в первую очередь его гибкостью и эффективностью.

Гибкость заключается в том, что, записывая имитационную программу на языке C++, программист имеет в своем распоряжении всю мощь универсального языка программирования с его развитыми механизмами построения абстракций. Это оказывается важным при работе со сложными структурами данных и нетривиальными алгоритмами, которые встречаются при моделировании таких сложных систем, как ТКС.

Под эффективностью подразумевается возможность написания на языке C++ программ, которые обладают выразительностью исходного кода (т. е. позволяют записывать решение задачи на высоком уровне абстракции) и в то же время не уступают в быстродействии аналогичным программам, написанным на языках более низкого уровня (например, C). Иными словами, язык C++ позволяет свести к минимуму расходы, связанные с введением дополнительных абстракций и записью программы на более высоком уровне (abstraction penalty). Замедление программы в несколько раз допустимо, если время ее прогона измеряется секундами. В таких случаях производительностью можно пожертвовать в пользу, например, удобства разработки модели. Однако по мере усложенения модели все более недопустимым становится наличие abstraction penalty. Время имитации больших ТКС измеряется часами, поэтому быстродействие имитационной программы играет важную роль.

Язык C++ в силу своей универсальности не предоставляет, в отличие от языков ИМ или пакетов

■ *Таблица. Библиотеки обобщенного программирования на языке C++*

Предметная область	Библиотеки
Линейные последовательности (динамический массив, список, стек, очередь, куча, дека, сбалансированное дерево) и алгоритмы над ними	STL (Standart Template Library) [3]
Алгоритмы и структуры данных вычислительной геометрии	CGAL (Computational Geometry Algorithm Library) [4]
Графы и алгоритмы на графах	BGL (Boost Graph Library) [5]
Матрицы и матричные вычисления	MTL (Matrix Template Library) [6], GMCL (Generative Matrix Computational Library) [7]
Многомерные массивы для численных методов	blitz++[8]
Итеративные численные методы	ITL++ (Iterative Template Library)[9]

ИМ, готовых средств для непосредственной записи имитационной программы. Поэтому при написании имитационной программы на C++ программисту необходимо либо самому реализовывать инфраструктуру ИМ, обеспечивающую взаимодействие компонент модели и проведение вычислительного эксперимента (например, календарь событий), а также многие вспомогательные компоненты, которые часто используются при построении моделей (например, генераторы случайных чисел с заданными законами распределения), либо прибегать к использованию этих компонент, уже кем-то реализованных и оформленных в виде библиотек.

Часто вместе с библиотекой имитационного моделирования поставляются программы, облегчающие проведение имитационных экспериментов, составляя вместе систему или пакет имитационного моделирования. Например, в состав пакета OMNeT++ [1] входят визуальный редактор моделей gned, среда проведения имитационных экспериментов tkenv, средства для анализа результатов экспериментов и т. п. Вместе с системой имитационного моделирования зачастую поставляется библиотека имитационных моделей. Например, библиотека ns-2 [2] поставляется вместе с большим числом моделей каналов связи, узлов ТКС, работающих согласно различным протоколам.

Разработчики библиотек помимо выполнения главной задачи – предоставления некоторой полезной функциональности – стремятся сделать компоненты библиотеки пригодными для использования при решении как можно большего спектра задач, расширяя таким образом область применимости библиотеки и, следовательно, ее полезность. С пригодностью компоненты к ее повторному использованию связана возможность для пользователя настраивать ее поведение для своих нужд, а также способность быть использованной в различных окружениях. Основной прием достижения этих качеств – абстрагирование, которое в упрощенном виде можно описать как вынесение из компоненты всего, что не относится к ее сути, в виде параметров компоненты. Связывая с параметрами компоненты разные значения, можно менять ее поведение. Основной метод подобной параметризации компонент в ООП – использование механизма динамического полиморфизма, поддержка которого во многих ОО языках программирования реализована через механизм виртуальных функций.

К сожалению, подобную параметризацию средствами ООП можно проводить только до определенного предела, когда накладные расходы, связанные с использованием виртуальных функций, станут неприемлемо большими. Кроме урона быстродействию выполняемой программе использование динамического полиморфизма обладает рядом отрицательных эффектов, среди которых можно указать следующее:

1) уменьшение возможностей для статического контроля типов, что отодвигает момент диагностики неправильного использования компоненты на более поздний период: с момента компиляции на момент выполнения программы;

2) необходимость наследования классов, которые предполагается использовать как параметры компоненты, от определенных базовых классов, что снижает адаптируемость компоненты.

В следующем разделе будут более детально обсуждаться препятствия на пути использования методов ООП для повышения уровня повторного использования компонент библиотеки.

Указанные проблемы успешно решаются средствами обобщенного программирования, которое поддержано в языке C++ мощным механизмом шаблонов. Библиотеки, выполненные в парадиг-

ме обобщенного программирования, сочетают широкие возможности настройки компонент под нужды пользователя, удобство их использования с эффективностью результирующего кода.

В таблице перечислен ряд предметных областей, для которых созданы и применяются библиотеки обобщенного программирования на C++.

За последнее десятилетие обобщенное программирование оформилось в поддисциплину информатики, «которая занимается поиском абстрактных представлений эффективных алгоритмов, структур данных и других понятий программного обеспечения вместе с их систематической организацией. Целью обобщенного программирования является выражение алгоритмов и структур данных в форме, которая обеспечивает легкость их адаптации, возможность взаимодействия (interoperability) между ними, а также позволяет прямое их использование при построении программного обеспечения» [10].

В обобщенном программировании конфигурирование компонент (т. е. связывание с параметрами компонент определенных значений) происходит в момент компиляции программы, а не ее выполнения, как в объектно-ориентированном программировании. При этом используются языковые механизмы (шаблоны классов и функций), имеющие другие характеристики для построения адаптируемых компонент, нежели механизмы ООП (виртуальные функции). Это определяет необходимость при проектировании в обобщенном стиле следования принципам, в значительной степени отличающимся от принципов объектно-ориентированного проектирования[1].

В данной работе рассматривается применение методов обобщенного программирования при разработке библиотеки имитационных моделей с тем, чтобы придать ей следующие качества: модульность, эффективность, ранняя диагностика неправильного использования модулей, расширяемость, адаптируемость к окружению, выразительность исходного кода. Рассмотрим наиболее важные понятия компонентного подхода к созданию имитационных моделей на примере библиотеки OMNeT++ [1].

Модель в OMNeT++ состоит из иерархически вложенных блоков-модулей, вложенность блоков при этом не ограничена, что позволяет пользователю создавать модели сложных иерархических систем.

Каждый модуль может рассматриваться как относительно независимая сущность, которая взаимодействует с другими модулями, как правило, только при помощи механизма посылки сообщений. Посылка сообщения модулем заключается в помещении объекта, представляющего сообщение, в один из выходных портов (output gate), ассоциированных с модулем. Они абстрагируют модуль от приемника сообщения, позволяя таким образом установить сообщение между любыми двумя модулями, имеющими порты. Серьезным недостатком такого подхода является невозможность выразить синтаксически (и проверить на этапе компиляции), что модуль может принимать сообщения только определенного типа.

С каждым модулем могут быть ассоциированы параметры, при помощи которых можно настраивать его поведение. Параметр может иметь тип только из предопределенного набора типов, что снижает выразительность описания модели. Параметры модуля хранятся как набор пар «ключ–значение», где ключом является строка, а значением – variant-подобный тип. Такой подход не предоставляет возможности компилятору проверить, что среди параметров модуля есть определенный параметр и он обладает нужным типом. Кроме этого доступ к параметрам сопряжен с высокими накладными расходами.

В OMNeT++ модули бывают двух типов: простые и составные. Простые модули содержат алгоритм поведения некоторого элемента модели. Составные модули агрегируют в себе другие модули (неважно, простые или составные), соединяют их входные и выходные порты, а также связывают с их параметрами определенные значения. Существенным моментом здесь является то, что простые модули берут на себя всю поведенческую составляющую модели (все поведение модели есть композиция поведения простых модулей), а составные модули – всю организационную составляющую модели (именно они задают, как модули соединяются и какие значения будут присвоены их параметрам). Моделью является составной модуль верхнего уровня, т. е. такой, который не принадлежит никакому другому модулю.

Обобщенное программирование является подходящей технологией для реализации простых модулей, поскольку позволяет создавать сильно параметризованные, обобщенные компоненты, которые не уступают в эффективности непараметризованным модулям.

К сожалению, методов обобщенного программирования недостаточно для разработки всей библиотеки имитационных моделей – кроме высокопараметризованных простых модулей необходимо предоставить средства, которые помогают пользователю отобразить замысел конструируемой модели в набор классов и их параметров, который реализует задуманную модель. Эта проблема решается средствами производящего программирования (generative programming) [14] и выходит за рамки данной статьи.

В OMNeT++ для описания составных модулей (и интерфейса простых модулей для того, чтобы они могли быть частью составного модуля) используется специальный язык NED (Network

[1] Для знакомства с ними можно порекомендовать работы [11–13].

Description Language). Компилятор с языка NED преобразует описания модулей в исходный код на C++, который после обработки компилятором C++ компонуется вместе с объектными файлами, содержащими алгоритмы модели, формируя таким образом имитационную программу. Именно здесь становится важной разница между простыми и составными модулями: простые модули лучше всего записывать на алгоритмическом языке программирования, а их конфигурирование лучше поручить специальному генератору.

Генератор составных компонент может быть реализован и в рамках языка C++ при помощи шаблонного метапрограммирования [15]; перспективность этого подхода в настоящее время исследуется автором статьи.

В данной работе представлен подход к созданию модулей имитационных моделей как обобщенных компонент, оценивается адаптируемость реализованных таким образом простых модулей, использованные техники реализации сравниваются с альтернативами.

Автором статьи разработана и развивается библиотека имитационного моделирования tksym, которая включает библиотеку имитационных моделей узлов и каналов связи ТКС, используемую для сравнительного анализа различных алгоритмов маршрутизации в ТКС. Простые модули реализованы в ней при помощи описываемых ниже техник обобщенного программирования. В настоящее время для составления из простых модулей имитационной модели программист должен написать код, явно их конфигурирующий, что весьма неудобно; поэтому очень актуальна задача автоматической генерации модели по высокоуровневой спецификации.

Сравнение механизмов статического и динамического полиморфизма в языке C++

Основной метод достижения гибкости программных продуктов – разбиение на модули, при котором стремятся, с одной стороны, ослабить зависимости между модулями (слабое зацепление между модулями), а с другой – составлять модули из сильно связанных элементов (сильная связность в модулях).

В языке C++ для уменьшения зависимости между модулями можно использовать механизм виртуальных функций (который обеспечивает полиморфизм времени выполнения, или динамический полиморфизм) или механизм шаблонов языка C++ (который обеспечивает полиморфизм времени компиляции, или статический полиморфизм). Первый подход лежит в основе объектно-ориентированного программирования, второй – в основе обобщенного программирования. Каждый из них имеет свои преимущества и недостатки, которые в зависимости от специфики разрабатываемого программного обеспечения проявляются в разной степени.

Придание гибкости программе при помощи механизма виртуальных функций часто имеет положительный эффект, что широко проиллюстрировано в литературе по объектно-ориентированному программированию. Хорошее изложение приемов решения типичных задач проектирования (паттернов проектирования) в рамках парадигмы объектно-ориентированного программирования читатель может найти в классическом труде [16].

К сожалению, есть ситуации, когда от применения механизма виртуальных функций приходится отказаться по ряду причин.

Основная причина – ухудшение производительности программы по сравнению с аналогичной по функциональности программой, но написанной без разбиения на модули и уменьшения зависимостей между модулями. Этот эффект известен как abstraction penalty – накладные расходы, связанные с введением абстракций.

Ухудшение производительности в первую очередь обусловлено неспособностью компилятора проводить оптимизации кода вокруг точки вызова виртуальной функции, поскольку он не может определить, какая именно функция будет вызвана. Кроме этого, на некоторых процессорных архитектурах косвенный вызов, к которому сводится вызов виртуальной функции, сбрасывает содержимое конвейера команд, что также наносит удар по производительности.

Урон производительности, причиняемый использованием механизма виртуальных функций, зависит от размера функций и частоты их вызова. Чем меньше время выполнения виртуальной функции и чем чаще она вызывается, тем больше урон. Например, если спроектировать библиотеку матричных вычислений так, что различные классы матриц должны переопределить виртуальную функцию доступа к элементу матрицы, объявленную в базовом для всех матриц классе абстрактной матрицы, то эффективность алгоритмов, записанных в терминах абстрактного базового класса всех матриц, может снизиться во много раз по сравнению с алгоритмами, работающими над конкретными типами матриц.

Если функция достаточно большая и не может быть встроена в точку вызова, отрицательный эффект от виртуального вызова достаточно мал, и с ним можно смириться в пользу большей гибкости программы и более быстрой ее компиляции.

При интенсивном использовании виртуальных функций иногда наблюдается тенденция к созданию большого числа маленьких объектов в свободной памяти. Если не применять оптимизированного на работу с маленькими объектами менеджера памяти, их размещение может потребовать излишнего расхода памяти и времени.

Использование абстрактных базовых классов уменьшает возможности статического контроля типов. Показательным примером может служить проектирование контейнера элементов, подразуме-

вающее при таком подходе определение абстрактного базового класса для всех классов, которые могут храниться в контейнере. Назовем этот класс `Object`, а контейнер объектов типа `Object` – `ObjectContainer`. Допустим, что пользователь желает хранить в этом контейнере только объекты определенного типа `X`. К сожалению, это намерение при использовании `ObjectContainer` никак не может быть выражено синтаксически в коде – компилятор не выдаст сообщения об ошибке при попытке вставить в `ObjectContainer` объект другого класса. Кроме этого, после извлечения элемента из контейнера необходимо делать понижающее приведение типа от `Object` к `X` и проверять его успешность, что отрицательно сказывается на выразительности кода. (В данном случае проблему можно решить введением обертки над `ObjectContainer`, которая будет гарантировать типобезопасность. Однако такое решение очень плохо масштабируется и поэтому непригодно в качестве универсального подхода).

Некоторые авторы [17] указывают еще ряд отрицательных эффектов, которые могут проявиться при попытках придания большей гибкости программной архитектуре при помощи механизма виртуальных функций, а именно: необходимость заблаговременного проектирования классов, учитывая возможные изменения в будущем (preplanning problem), введение избыточного числа дополнительных абстракций, наличие которых затрудняет понимание кода программы (confusion problem) и пр.

При использовании шаблонов языка C++ информация, которая помогает компилятору сгенерировать оптимизированный код, не теряется, и поэтому при использовании хороших оптимизирующих компиляторов получаемый код не уступает значительно по эффективности нешаблонному коду. Безопасность типов также сохраняется, поскольку реальный тип объектов известен в момент компиляции.

Слабыми сторонами шаблонного программирования на C++ считаются следующие.

1. Недостаточная спецификация синтаксических ограничений на параметры шаблона. Если при программировании с виртуальными функциями абстрактный базовый класс является тем местом, где сформулированы синтакисические требования класса к своему параметру (наличие методов с определенной сигнатурой), то аналогичного способа спецификации требований к параметрам шаблона в языке C++ нет – приходится использовать дополнительные техники вроде concept checking [18] для отражения в исходном коде требований, которым должны удовлетворять аргументы шаблона.

2. Неосторожное использование шаблонов может привести к значительному увеличению объема исполняемого кода – так называемому «разбуханию» кода. Существуют несложные приемы, позволяющие избежать подобного эффекта.

3. Шаблонная программа может компилироваться и компоноваться значительно дольше, чем аналогичная программа без использования шаблонов. Кроме этого, использование шаблонов уменьшает возможности использования раздельной компиляции.

Некоторые понятия обобщенного программирования

Важнейшим приемом обобщенного проектирования является выделение минимальных требований компоненты к своим параметрам и реализация компоненты с минимальными предположениями о своих параметрах. Чем более слабые ограничения накладываются на параметры компоненты, тем больше компонент могут стать ее аргументами. Требования усиливаются, если это позволяет сделать компоненту более эффективной. Разные алгоритмы, выполняющие одну задачу, могут сосуществовать, если они работают в различных предположениях о своих параметрах; при этом предоставляются средства автоматического выбора наиболее подходящего алгоритма под заданный набор параметров.

Наборы требований, предъявляемых обобщенной компонентой к своим параметрам, называют *концепциями*. Требования делятся на синтаксические и семантические.

Синтаксические требования могут быть проверены на этапе компиляции – при попытке использовать компонент с параметром, не удовлетворяющим его синтаксическим требованиям, будет выдана ошибка компиляции. В ООП синтаксическим требованием к параметру класса является свойство «быть наследником определенного базового класса». Это подразумевает, что в аргументе компоненты должны быть объявлены функции-члены, сигнатура которых в точности совпадает с сигнатурой функций-членов базового класса[1].

В обобщенном программировании синтаксическое требование выражается менее жестко: вместо требования совпадения сигнатуры функции-члена требуется, чтобы она имела параметры, к типам которых может быть приведен тип определенного выражения (т. е. требования приводимости).

Семантические требования не могут быть проверены на этапе компиляции (например, что некоторая последовательность отсортирована). Некоторые из них могут быть выражены в коде в виде утверждений о поведении параметра компонента, некоторые представляют собой утверждения о вычислительной сложности той или иной операции: например, на являющийся параметром шаблона контейнер может быть наложено требование выполнять вставку элемента в начало контейнера за время O(1) (этому требованию удовлетворя-

[1] Ослаблением этого правила в C++ являются ковариантные типы возвращаемых значений.

ют шаблоны классов `std::list`, `std::deque` и не удовлетворяют `std::vector`, `std::set`).

Тип, удовлетворяющий набору требований концепции, называется *моделью* этой концепции (*моделирует* концепцию). Концепция, которая расширяет набор требований другой концепции, называется ее *уточнением*.

Проектирование простого модуля в обобщенном стиле

Рассмотрим различные подходы к реализации часто встречаемой в имитационных программах модели устройства (которая является несколько упрощенным аналогом модуля `Server` системы ИМ Arena [19]), работающего по следующему алгоритму.

Когда в устройство поступает объект для обработки *e*, проверяется, не обрабатывает ли устройство другой объект. Если да, то пришедший объект *e* сохраняется в ассоциированной с устройством очереди. Если нет, вычисляется время *t*, которое займет обработка *e*, и устройство переходит в состояние обработки *e* на время *t*. После окончания обработки *e* посылается далее, устройство спрашивает у очереди очередной объект для обработки и, если таковой имеется, приступает к его обработке, если нет, то переходит в состояние ожидания.

В библиотеке tksym модель устройства реализована следующим образом:

```
template
<
   class Base,
   class World,
   class Queue,
   class ProcessingTime,
   class Sink,
   class Events
>
struct Server : Base, World, Queue, ProcessingTime, Sink, Events
{
   typedef typename Base::Entity   Entity;

   using Queue::queue;
   using Events::events;

   // "входной порт" модуля, через который
   // он получает сообщения от других модулей
   void process(Entity e)
   {
      if (being_sent_)          // если устройство занято,
           queue().push(e);     // сохраним e в очереди
      else                      // иначе
           startProcessing(e);// начнем его обработку
   }

   Entity const & beingSent() const{ return being_sent_; }

private:
   void startProcessing(Entity e)
   {
       // оценим время t, которое займет обработка e
       // и запланируем вызов метода "release" через t секунд
      World::schedule(ProcessingTime::get(e),
          boost::bind(&ResourceEx::release,this));

      // перейдем в состояние "занято"
      being_sent_ = e;

      // оповестим подписавшихся на прослушивание событий,
      // что обработка e началась
       events().OnStartProcessing(e);
   }

   void release()
   {
      // поместим обработанный объект в "выходной" порт
      Sink::process(being_sent_);

      // оповестим слушателей событий,
      //что обработка закончилась
      events().OnStopProcessing(being_sent_);

      // перейдем в состояние "ожидание"
      being_sent_ = Entity();

      // если в очереди есть еще объекты для обработки
      if (!queue().empty())
      {
           // выберем из них один
           // и приступим к его обработке
           startProcessing(queue().top());
           // удалим обрабатываемый объект из очереди
           queue().pop();
      }
   }

private:
   Entity being_sent_;
};
```

Рассмотрим характерные черты представленного дизайна класса `Server`.

Компонент наследуется от набора параметров шаблона (которые иногда называются стратегиями – policy [12]) и реализуется с использованием исключительно имен, зависимых от параметров шаблона. Это позволяет использовать класс `Server` в любом окружении – достаточно лишь связать его параметры с типами, которые удовлетворяют определенному набору требований.

Примером синтаксических требований может служить требование от параметра `Base` иметь внутренний тип `Entity`, который обладает конструктором по умолчанию, конструктором копирования и оператором приведения к типу, который может быть условием в инструкции `if`. Будем считать, что этот оператор приводит объект типа `Entity` к типу `bool`. Класс `Server` считает, что результат такого приведения равен `false` тогда и только тогда, когда `Entity` сконструирован по умолчанию (что может служить примером семантического требования). Сконструированный по умолчанию `Entity` служит для обозначения состояния устройства «свободен».

Если стратегия предоставляет несколько функций-членов, то они объединяются в один класс (будем называть такие стратегии составными). Это упрощает создание стратегий, вызовы к себе делегирующих другим классам. В нашем примере та-

кими стратегиями являются параметры Queue и Events. Если бы 4 функции-члена очереди не были агрегированы в объект, возвращаемый функцией queue(), их пришлось бы реализовывать к каждом классе, связываемом с параметром Queue.

Для доступа к методам обычных стратегий используется запись Стратегия::Метод(). Для доступа к методам составной стратегии Стратегия, метод доступа к ним стратегия() вносится в область имен модуля (при помощи using-объявления), после этого доступ к методу Стратегия::методА() осуществляется как стратегия().методА().

Альтернативный подход мог бы заключаться в доступе к методам стратегий через указатель this: this->МетодНекоторойСтратегии(). Однако при его использовании возможны неоднозначности поиска имен в базовых классах. Например, параметр Sink мог быть тоже унаследован от типа, являющегося параметром World. В этом случае два базовых класса содержали бы метод schedule, что привело бы к ошибке компиляции.

Среди параметров модуля выделяется класс Base, при помощи которого можно достичь так называемой управляемой виртуальности методов модуля [13]. Например, если бы в классе Base была объявлена виртуальная функция-член void process(Entity e), то и функция-член void Server::process(Entity e) тоже стала бы виртуальной.

Класс Server позволяет другим классам инспектировать свое состояние. Для этого он предоставляет метод, позволяющий прочитать его состояние (beingSent), а также события о изменении состояния. Слушатель этих событий задается параметром Events.

Каждый из параметров модуля может быть реализован разными способами.

Например, класс Server можно использовать с различными очередями. Они могут различаться дисциплиной обслуживания: простые очереди по принципу first come – first served (FCFSQueueHolder) или first come – last served (FCLSQueueHolder), очереди с приоритетом, для которых пользователь может указать различные предикаты, упорядочивающие объекты типа Entity (PriorityQueueHolder).

Реализация классов FCFSQueueHolder, FCLSQueueHolder и PriorityQueueHolder выглядит следующим образом:

```
template <class QueueStorage>
  struct QueueHolder
{
  typedef QueueStorage      queue_type;

  queue_type  const & queue() const { return queue_; }
  queue_type        & queue()       { return queue_; }
private:
  queue_type          queue_;
};

template <class Entity> struct FCFSQueueHolder
    : QueueHolder <std::queue<Entity> >
{};

template <class Enity>  struct FCLSQueueHolder
    : QueueHolder <std::stack<Entity> >
{};

template <class Entity, class Comparator>
    struct PriorityQueueHolder :
      QueueHolder <
            std::priority_queue<Entity, Comparator> >
{};
```

Владение очередью может быть монопольным (классы, использующие QueueHolder) или разделяемым (QueueInDerived или QueueIndirect):

```
// Предполагаем, что у Derived есть метод
// QueueType & getQueue() и QueueInDerived является
// предком (возможно, непрямым) Derived
template <class QueueType, class Derived>
  struct QueueInDerived
{
  typedef QueueType   queue_type;

  queue_type const & queue() const {
    return static_cast<Derived const *>
                              (this)->getQueue(); }
  queue_type & queue(){
    return static_cast<Derived *>
                              (this)->getQueue(); }
};

template <class QueueType, class Holder>
  struct QueueIndirect
{
  typedef QueueType   queue_type;

  void setHolder(Holder * h) { holder_ = h; }

  queue_type const & queue() const
    { return holder_->queue(); }
  queue_type         & queue()
    { return holder_->queue(); }

private:
  Holder       * holder_;
};
```

Очередь может быть бесконечной емкости (как, например, FCFSQueueHolder) или иметь фиксированную емкость. В случае, если происходит попытка вставки объекта в переполненную очередь, возможны разные стратегии: игнорирование объекта, передача очередью объекта обработчику объектов, которым не хватило в очереди места, генерация исключения – все они могут быть отражены соответствующими классами.

Требования класса Server к платформе ИМ минимальны: от шаблонного параметра World требуется только наличие функции-члена schedule, первым параметром которой является тип, представляющий время (к нему должен приводиться тип, возвращаемый функцией ProcessingTime::get()), а вторым параметром – тип, к которому можно привести функционал, имеющий ноль аргументов

(в текущей версии библиотеки tksym для этого используется тип boost::function<void ()>).

Вместо наследования модуля от стратегий можно было бы использовать агрегирование стратегий, что, однако, имеет недостатки по сравнению с наследованием по следующим показателям.

1. Экономия памяти. Каждый агрегируемый класс (даже, если он имеет нулевой размер) вносит ненулевой вклад в размер объекта класса Server. Наследование от класса нулевого размера не увеличивает размер наследника (если компилятор поддерживает оптимизацию пустого базового класса, Empty Base Class Optimization – EBCO).

2. Доступ к наследнику. Параметризация базового класса своим наследником является мощным и распространенным приемом, известным как модель странного рекуррентного шаблона (curiously recurring template pattern, CRTP)[13]. Базовый класс может получить доступ к методам своего наследника приводя указатель this к типу указателя на своего наследника. При этом не вносятся накладные расходы ни по времени выполнения, ни по используемой памяти. Примером использования такого приема может служить класс QueueInDerived. Агрегированный класс не имеет доступа к содержащему его классу («хозяину») (чтобы обойти это, мы могли бы запоминать в агрегируемом классе указатель на его «хозяина», что увеличило бы расход памяти, или при вызове методов агрегированного объекта передавать ему указатель на «хозяина», что не всегда удобно).

Альтернативой наследованию модуля от классов стратегий могло бы быть применение для класса модуля CRTP: класс Server параметризовался бы классом наследником. При помощи этого приема элегантно реализуется архитектура, основанная на примесях (mixin-based design).

Допустим, у нас есть набор классов-примесей M_1, ..., M_n, от которых наследуется некоторый класс C. Некоторые примеси зависят от других примесей. Проблема заключается в организации доступа из одной примеси к методам другой примеси. Если не использовать шаблонов, такой доступ можно реализовать, заведя виртуальный базовый класс с методами, к которым нужно организовать доступ, и наследования от него всех примесей, что является довольно хрупким решением. Другой способ – использование динамического приведения типа от указателя this примеси к нужной примеси. Недостатком такого решения является потеря статической типизации. Неудачное приведение типа может быть обнаружено только в момент выполнения программы. Параметризация классов-примесей классом C является элегантным решением этой проблемы: класс-примесь может получить доступ к методам других примесей, приведя указатель this к указателю на класс C (поскольку класс C унаследован от классов M_1, ..., M_n, то через класс C можно получить доступ к методам примесей).

В ранних версиях библиотеки tksym основным способом параметризации компонент модели было использование приема CRTP. Оказалось, что применительно к построению библиотеки моделей этот прием обладает следующими недостатками.

1. Методы базового класса могут вызывать методы производного класса, однако, к сожалению, при определении базового класса нельзя использовать типы, определяемые в производном классе. Для обхода этой проблемы типы, необходимые базовому классу, приходилось ему передавать как параметры шаблона, что в некоторых случаях загромождало исходный код.

2. Прием CRTP плохо масштабируется. По мере роста класса C увеличивается вероятность того, что найдутся две его примеси, которые содержат метод с одним и тем же именем. Если третья примесь обратится к этому методу, компилятор не сможет разрешить неоднозначность. Для разрешения этой неоднозначности может потребоваться введение прокси-классов, что при наличии в классах-примесей большого числа методов весьма неудобно.

Заключение

Архитектура рассмотренного выше класса Server является адаптацией для нужд библиотеки имитационных моделей архитектуры, основанной на стратегиях (policy-based design), описанной в работе [12].

Высокопараметризованный модуль Server, будучи инстанциирован с определенным набором параметров, не уступает по эффективности (в терминах времени выполнения и занимаемой памяти) непараметризованному монолитному аналогу при использовании современных компиляторов языка C++ (эксперимент проводился с использованием компиляторов Microsoft Visual C++ 7.1 и Intel C++ 8.0). К сожалению, за абстрактность кода и его эффективность приходится платить временем компиляции, однако есть надежда, что по мере развития технологии компиляции эта проблема станет менее острой, чем в настоящее время.

Модуль Server обладает высокой способностью к повторному использованию за счет высокой параметризации и того, что реализация класса Server не зависит от имен, независимых от параметров шаблона, т. е. в реализации класса отсутствует какая-либо информация, зависимая от контекста определения класса Server. Проблема несоответствия компоненты, предназначенной быть аргументом модуля, синтаксическим требованиям, предъявляемым модулем своему параметру, легко решается введением «обертки» над компонентом, адаптирующей его интерфейс.

Среди программистов бытует выражение, что «любую проблему программной архитектуры можно решить при помощи введения дополнительного слоя абстракции. Единственная проблема, не поддающаяся такому решению, – наличие слишком большого числа слоев абстракции». При

разработке библиотеки крайне важно сделать интерфейсы компонент и их требования к параметрам согласованными, чтобы свести к минимуму необходимость введения адаптеров. Это способствует лучшему изучению библиотеки, более глубокому ее пониманию. Несоответствие интерфейсов не является фатальным, однако избыточное введение адаптеров представляется попыткой исправить архитектурные ошибки.

При проектировании интерфейса модуля и спецификации требований на его параметры следует стремиться достичь баланса между простотой интерфейсов и тем, чтобы через них можно было получить достаточное для эффективой реализации модуля и работы с ним количество информации. Например, для того, чтобы пользователи класса `Server` могли эффективно инспектировать его состояние, класс `Server` позволяет подписаться на прием сообщений об изменении своего состояния.

Укажем направления будущих работ.

1. Разработка принципов создания составных модулей. Поскольку их основная задача – агрегировать в себе другие компоненты, соединить их друг с другом и связать с их параметрами определенные значения, представляется перспективной реализация составных модулей как классов метафункций (metafunction class) с использованием библиотеки Boost.Mpl [15].

2. Средства высокоуровневой спецификации (как в текстовом, так и в графическом виде) модели пользователем, которому достаточно функциональности существующих модулей. Средства трансляции высокоуровневого описания модели в набор классов языка C++. Средства, описывающие такому транслятору интерфейс модулей.

3. Реализация инструментов, облегчающих проведение вычислительных экспериментов. При решении этой задачи важно максимально использовать уже существующие аналогичные инструменты, программируя только переходники (bridge), обеспечивающие взаимодействие с ними модулей библиотеки.

Литература

1. **OMNeT++** Home page. http://www.omnetpp.org/
2. **The Network** Simulator - ns-2. Home page. http://www.isi.edu/nsnam/ns/
3. **Standard** Template Library Programmer's Guide http://www.sgi.com/tech/stl/
4. **Computational** Geometry Algorithm Library Home Page. http://www.cgal.org
5. **Boost** Graph Library Home Page http://www.boost.org/libs/graph/doc/table_of_contents.html
6. **Matrix** Template Library Home Page http://www.osl.iu.edu/research/mtl/
7. **Generative** Matrix Computational Library Home Page http://www-ia.tu-ilmenau.de/~czarn/gmcl/
8. **Blitz++** Home Page http://www.oonumerics.org/blitz/
9. **Iterative** Template Library Home Page http://www.osl.iu.edu/research/itl/
10. http://www.cs.rpi.edu/~musser/gp/dagstuhl/gpdag_2.html
11. **Austern M.** Generic Programming and the STL: Using and Extending the C++ Standard Template Library. – Addison-Wesley Professional; 1998. 576 p.
12. **Alexandrescu A.** Modern C++ Design: Generic Programming and Design Patterns Applied. – Addison-Wesley Professional, 2001. 352 p.
13. **Vandervoorde,D., Josuttis N.** C++ Templates: The Complete Guide. – Addison-Wesley Professional, 2002. 552 p.
14. **Czarnecki K., Eisenecker U.** Generative Programming: Methods, Techniques, and Applications. – Addison-Wesley, 1999. 864 p.
15. **Boost** Metaprogramming Library Home Page. http://www.boost.org/libs/mpl/doc/index.html
16. **Gamma E., Helm R., Johnson R., Vlissides J.** Design Pattern. – Addison-Wesley Professional; 1995. 416 p.
17. **Subject-oriented** programming Home Page. http://www.research.ibm.com/sop/
18. **The Boost** Concept Check Library. http://www.boost.org/libs/concept_check/concept_check.htm
19. http://www.arenasimulation.com/

НИКЛАУС ВИРТ – ПОЧЕТНЫЙ ДОКТОР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Язык должен быть очевидным и естественным отражением фундаментальных и наиболее важных концепций алгоритмов.

Н. Вирт

Приезд Никлауса Вирта в Россию приурочен к празднованию 250-летия МГУ и 150-летия швейцарского ETN (Eidgenoessische Technische Hochschule) – Высшей Политехнической школы в Цюрихе.

Однако этот визит начался с посещения Санкт-Петербурга, и это не случайно – у нашего Великого города давние научные связи со Швейцарией. Еще в XVIII веке в Санкт-Петербурге жил и работал великий Леонард Эйлер, похороненный в Некрополе Александро-Невской лавры, именем которого Вирт назвал свой первый язык. В Санкт-Петербурге в течение уже многих лет работает Международный математический институт им. Л. Эйлера.

Организатором приема профессора Н. Вирта, а также его соавтора и единомышленника профессора Юрга Гутхнехта в Санкт-Петербурге стала лаборатория «Технологии программирования» Санкт-Петербургского государственного университета информационных технологий, механики и оптики (СПбГУ ИТМО) и Санкт-Петербургский филиал корпорации *Borland*. Эти организации многим обязаны Н. Вирту, разработавшему в 1970 г. язык *Паскаль*, применение которого обеспечивает «дисциплину программирования»[1]. Несмотря на возможность использования на командном студенческом чемпионате мира по программированию ACM ICPC (Association for Computing Machinery Internatinal Collegiate Programming Contest) таких языков, как C/C++ и Java, команды СПбГУ ИТМО успешно применяют именно язык Паскаль.

При этом отметим, что по результатам последнего десятилетия СПбГУ ИТМО занимает второе место среди тысяч принимающих участие в чемпионате мира университетов[2], а в 2004 г. команда университета ИТМО стала чемпионом мира[3]. Возможно, это связано с тем, что в слове *алгорИТМОв* присутствует название нашего университета? Не следует также забывать, что в 2000 и 2001 гг. чемпионами мира была команда Санкт-Петербургского государственного университета (СПбГУ), которая в рейтинге за последнее десятилетие занимает третье место в мире. Это «способствовало тому, что центр подготовки лучших программистов страны сместился из Новосибирска и Москвы в Санкт-Петербург»[4].

Для корпорации *Borland* язык *Паскаль* – это их «все», так как практически любой программист мира, по крайней мере, слышал о таких продуктах корпорации, как *Turbo Pascal*, *Borland Pascal*, *Delphi*. Последний из этих продуктов, который создан 10 лет назад и непрерывно совершенствуется, является самым востребованным в России. На протяжении последних лет он неизменно занимает верхнюю строчку в рейтинге «Лучший продукт года» – ежегодном опросе читателей журнала «Мир ПК».

СПбГУ ИТМО и *Borland* уделяют большое внимание школьному образованию. Поэтому визит Н. Вирта 13 сентября 2005 г. начался с посещения физико-математического лицея № 239, который Дж. Сорос назвал лучшей школой мира. Ее окончили, кстати, не только А. В. Иванов (директор Санкт-Петербургского филиала корпорации *Borland*) и профессор В. Г. Парфенов (декан факультета информационных технологий и программирования СПбГУ ИТМО, лауреат премии Президента РФ в области образования), но двое из наших чемпионов мира – С. Оршанский и Д. Павлов, и все чемпионы мира из СПбГУ – А. Лопатин, Н. Дуров, О. Етеревский и В. Петров. Кроме того, ежегодно в каждой группе на кафедре компьютерных технологий СПбГУ ИТМО примерно половина студентов – вы-

[1] Дейкстра Э. Дисциплина программирования. М.: Мир, 1978. http://lib.ru/CTOTOR/DEJKSTRA

[2] Богатырев Р. Нас не догонят? Триумф России и провал США //Мир ПК. – 2005. – № 5. – С. 60–67. http://is.ifmo.ru/belletristic/_acm2005.pdf

[3] Богатырев Р. К истории чемпионатов мира ACM по программированию //Мир ПК. – 2004. – № 7. – С. 48–51. http://is.ifmo.ru/belletristic/_acmhist.pdf

[4] http://www.oberon2005.ru

пускники этого лицея. Встреча с Н. Виртом состоялась утром, в актовом зале школы. Учащихся было столько, что, как говорят в таких случаях, «не сидели только на люстрах». Классика программирования встретили овацией и стоя, а после его выступления так же проводили. Примерно пятнадцать минут Вирт уделил раздаче автографов на русских изданиях своих книг. «Эквакуировать» профессора из школы удалось не без значительных усилий со стороны руководителей приема.

После этого Н. Вирт и Ю. Гутхнехт прибыли в СПбГУ ИТМО, где состоялось расширенное заседание Ученого совета, на котором Н. Вирту вручался диплом и мантия почетного доктора университета ИТМО. 28 июня 2005 г. по моему представлению за это единогласно проголосовали все члены Совета.

Проректор университета профессор Ю. Л. Колесников представил гостям профессора В. Г. Парфенова, а также В. Л. Макарова – Президента Российской национальной ассоциации компаний – разработчиков ПО, в которую входят около ста программистских организаций, Т. А. Павловскую и А. А. Шалыто – профессоров СПбГУ ИТМО, А. С. Станкевича – ассистента кафедры компьютерных технологий, лауреата Премии Президента РФ в области образования, полученную им за подготовку команд и личное участие в чемпионатах мира по программированию, П. Маврина – чемпиона мира 2004 г., золотого медалиста чемпионата 2005 г., лауреата Премии Президента РФ, которую он получил за блестящие результаты, достигнутые на международной олимпиаде школьников по информатике.

После знакомства, фотографирования и получения автографов на книгах Н. Вирта и книгах о его языках все перешли в актовый зал, в котором количество людей, особенно молодых, нас поразило. Нечто подобное в стенах вузов я видел лишь однажды в 60-х годах, когда в ЛЭТИ, где я учился, приезжал Владимир Высоцкий. Снова, как и в школе, Н. Вирта встретили овацией и стоя.

В зале присутствовало огромное число студентов не только нашего вуза, но и многих других университетов Санкт-Петербурга. Люди стояли в проходах и у сцены.

Встреча открылась кратким приветственным словом профессора Ю. Л. Колесникова. После этого прозвучал гимн университета ИТМО, и мне было предоставлено слово для сообщения о жизни и творчестве Н. Вирта.

В начале своего выступления я сказал, что 28 июня, представляя Н. Вирта Ученому совету, я пояснял для не специалистов в области информатики, кто такой Н. Вирт, проведя сравнение его с Рафаэлем. Однако мне представляется, что в живописи титанов было значительно больше, чем в информатике. Об этом, в частности, свидетельствует галерея портретов основоположников информатики, опубликованная в этом году в нескольких номерах газеты для учителей «Информатика» (по два портрета в каждом номере). В этой галерее есть и портрет Н. Вирта.

После этого я кратко перечислил основные вехи биографии Н. Вирта. Родился 15 февраля 1934 г. в предместье Цюриха. В 1954 г. поступил в Высшую Политехническую школу в Цюрихе (ETH), в которой, говорят, учились и (или) работали 30 (!) Нобелевских лауреатов. Точно известно, что в этом университете учились такие гиганты науки, как А. Эйнштейн и Дж. фон Нейман. В 1958 г. Н. Вирт получил степень бакалавра по электротехнике; в 1960 г. – степень магистра в Лавальском университете г. Квебека (Канада); в 1963 г. – первый важный результат в информатике – создание на базе языка *Алгол* языка *Euler*, которой составил основу его докторской диссертации в университете Беркли (Калифорния США). После этого его пригласили в комитет IFIP по стандартизации *Алгола*. С 1963 по 1967 гг. Н. Вирт – доцент Стэнфордского университета в США; с 1968 г. – профессор компьютерных наук в ETH; в 1982–1984 гг. и 1988–1990 гг. возглавлял факультет компьютерных наук в ETH; с 1990 г. руководил Институтом компьютерных систем в ETH. 1 апреля 1999 г. Н. Вирт ушел с сохранением должностного оклада на пенсию по достижению 65-летнего возраста, так как ETH – вуз государственный, а для государственных служащих указанный срок предельный.

Основные достижения Н. Вирта в информатике:

1970 г. – создание языка *Паскаль* – первого в мире языка, в котором реализованы концепции структурного программирования;

1971 г. – публикация одной из основополагающих статей по структурному программированию «Разработка программы методом пошагового уточнения»[1], которое он создал вместе с теоретиками программирования Э. Дейкстром и Э. Хоаром[2].

1973 г. – участие в создании прообраза виртуальной машины для переноса языка *Паскаль* на разные платформы;

1976 г.– создание языка *Modula*, в котором были заложены основы мультипрограммирования;

1979 г.– создание языка *Modula-2*, поддерживающего концепцию модульности и квазипараллельных процессов;

1980 г. – создание 16-разрядного компьютера (!) *Lilith*, ориентированного на поддержку языка *Modula-2*;

1988 г. – создание языка *Oberon*, являющегося «минимальным» языком высокого уровня;

1996 г. – создание языка *Lola* для программирования настраиваемых схем (!).

Н. Вирт движется против течения – если у всех языки программирования со временем усложняются, то у него упрощаются: описание *Паскаля* занимало около 50 страниц, *Modula* – около 40, а Оберона – и вовсе 16... За это низкий поклон их автору.

Заслуги профессора Н. Вирта отмечены большим количеством наград, в том числе в 1984 г. премией

[1] http://www.acm.org/classics/dec95/

[2] Дал У., Дейкстра Э., Хоар Э. Структурное программирование. – М.: Мир, 1975.

Тьюринга, присуждаемой ACM за выдающиеся достижения в области информатики[1], а в 1988 г. – премией IEEE *Computer Pioneer*, которой среди 55 лауреатов также награждены и советские ученые В. М. Глушков, С. А. Лебедев и А. А. Ляпунов[2].

Н. Вирт – почетный доктор девяти университетов мира, в том числе такого известного, как университет Беркли (Калифорния, США), в котором он работал. Университет ИТМО – второй вуз в России, который присваивает Н. Вирту это почетное звание – в 1996 г. его избрали почетным доктором Новосибирского государственного университета.

В СССР и России книги Н. Вирта издавались неоднократно. Вот только некоторые из них:

- Систематическое программирование. Введение. М.: Мир, 1977.
- Паскаль. Руководство для пользователя и описание языка. М.: Финансы и статистика, 1982 (в соавторстве с К. Иенсен).
- Алгоритмы + структуры данных = программы. М.: Мир, 1985.
- Программирование на языке Модула-2. М.: Мир, 1987.
- Алгоритмы и структуры данных. СПб.: Невский диалект. 2001.

К визиту в нашу страну Н. Вирта выпущен диск, являющийся приложением к журналу «Мир ПК», 2005, № 9, который называется «От Паскаля к Оберону». Тираж диска более 50 тыс. экземпляров, что делает доступным творчество Н. Вирта для широкого круга лиц в нашей стране. На этом тематическом диске есть большой материал С. Оршанского (третьего участника звездной команды университета ИТМО) «О решении олимпиадных задач по программированию формата ACM ICPC»[3], который я «заставил» его написать.

На этом закончилось мое выступление, и слово было предоставлено А. В. Иванову, который достаточно подробно описал роль Н. Вирта и его языков в успехах корпорации *Borland*.

Потом выступил В. Л. Макаров, отметивший большое значение сегодняшнего события для всех присутствующих, особенно для молодых людей, которые со временем должны обеспечить лидерство России в области программирования в мире.

В. Г. Парфенов поблагодарил профессора Н. Вирта за язык *Паскаль*, который в «руках» студентов университета ИТМО стал сокрушительной силой на чемпионатах мира по программированию.

После этого П. Маврин и Д. Павлов вручили Н. Вирту футболку, в которой выступают участники чемпионатов мира по программированию.

И вот, наконец, наступил момент, когда под фанфары и овации зала профессор Ю. Л. Колесников вручил профессору Н. Вирту диплом и мантию Почетного доктора университета ИТМО. Видели бы вы, сколько студентов фотографировали этот момент!

После этого профессор Н. Вирт начал свое выступление и зачитал две страницы ... по-русски, рассказав, как он учил этот непростой язык.

Хочу отметить, что в ETH для профессоров компьютерных наук знание русского языка, видимо, становится необходимым: сменивший Н. Вирта профессор Бертран Мейер (Bertrand Meyer), создатель языка *Effel*, также знает русский язык, причем практически в совершенстве.

В дальнейшем прозвучала докторская лекция Н. Вирта на английском языке, поскольку аудитории, ввиду ее специфики, не потребовался перевод. По окончании лекции многие студенты получили автограф классика на его книгах.

В этот день в университете ИТМО классик был не только на сцене, но и в зале – Н. Вирта пришел слушать член-корреспондент РАН Юрий Владимирович Матиясевич, также выпускник 239 школы, который, будучи аспирантом Ленинградского государственного университета, решил десятую проблему Гильберта, о чем и было объявлено в зале.

Праздник закончился, но Н. Вирт продолжил работу – он и профессор Ю. Гутхнехт приняли участие в пресс-конференции, на которой я, в частности, спросил: «Почему язык *Oberon*, если он такой хороший, так мало известен?». На это профессор Ю. Гутхнехт не без иронии ответил: «Люди получают то, что заслуживают», – и предложил приз любому, кто предложит задачу, которую нельзя будет достаточно эффективно решить на этом языке.

В ответах на вопросы Н. Вирт отметил высокое качество образования в нашей стране, и особенно наличие элитных (в смысле высокого интеллекта учащихся и учителей) школ, которые, к примеру, в Швейцарии отсутствуют. Он также заметил, что все его разработки являются открытыми, так как университеты должны нести знания людям.

После завершения пресс-конференции у меня было некоторое время, чтобы рассказать профессорам Н. Вирту и Ю. Гутхнехту о предложенных мною автоматном программировании[4] и движении за открытую проектную документацию[5], но это уже совсем другая история.

Встреча не успела закончиться, а большой ее фрагмент показали по НТВ – визит Н. Вирта не только профессиональное, но и значительное общественное событие.

Шалыто А. А.,
доктор техн. наук, профессор, заведующий кафедрой «Технологии программирования» СПбГУ ИТМО.

[1] Лекции лауреатов премии Тьюринга за первые двадцать лет 1966–1985. М.: Мир, 1993.

[2] Шалыто А. А. У нас была Великая эпоха // Информационно-управляющие системы. – 2003. – № 1. – С. 52–56. http://is.ifmo.ru/belletristic/pre/

[3] http://is.ifmo.ru/works/orshanskiy

[4] Шалыто А. А., Шопырин Д. Г. Объектно-ориентированный подход к автоматному программированию // Информационно-управляющие системы. – 2003. – № 5. – С. 29–39. http://is.ifmo.ru/works/ooaut.pdf.

[5] Шалыто А. А. Новая инициатива в программировании. Движение за открытую проектную документацию / / Информационно-управляющие системы. – 2003. – № 4. – С. 52–56, http://is.ifmo.ru/works/open_doc/

ПАМЯТИ ПЕРОВСКОЙ ЕВГЕНИИ ИВАНОВНЫ

13 октября 1932 – 16 сентября 2005

Ушел из жизни замечательный педагог, видный ученый, обаятельная женщина профессор Перовская Евгения Ивановна. Она работала на кафедре вычислительных систем и сетей Санкт-Петербургского государственного университета аэрокосмического приборостроения (ГУАП) с момента ее основания, являясь одним из ведущих ее сотрудников.

Долгое время Евгения Ивановна была главным методистом кафедры и внесла большой вклад в подготовку инженеров по специальности «Вычислительные машины, комплексы, системы и сети». При ее непосредственном участии было подготовлено свыше 3500 специалистов. Все они помнят ее как прекрасного преподавателя и доброжелательного, отзывчивого человека.

Е. И. Перовская – известный ученый в области моделирования сложных систем, опубликовавшая свыше 200 научных трудов. Ее научные труды и лекции отличались ясностью системного мышления и широкими обобщениями.

Выпускница физико-механического факультета Ленинградского политехнического института, она начала свою трудовую деятельность в Институте электромеханики Академии наук СССР, где подготовила кандидатскую диссертацию. В 1965 году перешла на кафедру технической кибернетики Ленинградского института авиационного приборостроения (ныне ГУАП), где и работала до последних дней своей жизни.

Ее докторская диссертация была посвящена решению проблем автоматизации дискретных производственных процессов. В этот период был разработан язык для описания роботизированных производств LAROT, существенно упростивший программирование робототехнических комплексов, станков с программным управлением, транспортных систем и складского хозяйства.

Е. И. Перовская ввела понятие вариативности технологических процессов и связала его с понятием избыточности и многофункциональности. Она занималась вопросами календарного планирования и получила в этом направлении существенные новые результаты. Ею была предложена обобщенная модель задачи календарного планирования процессов мелкосерийного производства и разработан метод двойного ранжирования для решения возникающих при этом комбинаторных задач. Эти работы были особенно важны при создании гибких систем автоматизации. За разработку первого в мире гибкого автоматизированного цеха металлообработки на Днепропетровском электровозостроительном заводе Е. И. Перовская была удостоена премии Совета Министров СССР за 1983 год.

В последние годы Евгения Ивановна занималась разработкой теории сложных систем, включающих человека, технологию, культуру и природную среду. Создание целостного подхода для исследования функционирования таких систем стало особенно важным в наше время для обеспечения устойчивого развития человечества.

Е. И. Перовская была деятельным и разносторонним человеком. Ее излюбленным видом отдыха были байдарочные походы, к которым она приобщила многих друзей и учеников, а ее коллеги по Институту электромеханики помнят Евгению Ивановну как выдумщицу, организатора и обязательного участника веселых представлений, праздничных вечеров и балов.

Ее доброта и душевность согревали окружающих, к ней часто шли за поддержкой и советом в трудную минуту.

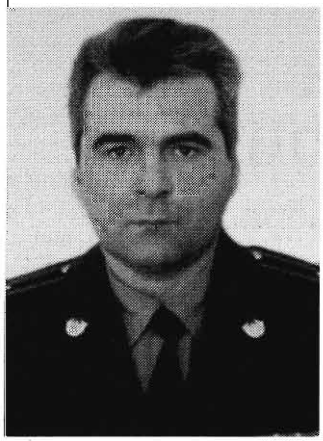
До последнего дня она вела активную научную и педагогическую деятельность, читая лекции и выступая с докладами на научных конференциях.

Добрый, отзывчивый и чуткий педагог Е. И. Перовская отдавала много времени подготовке научных кадров, щедро делясь богатейшим опытом и знаниями с молодыми учеными, аспирантами и докторантами. Коллеги и ученики Е. И. Перовской всегда будут помнить ее и по мере сил продолжать дело, начатое ею.

Коллектив кафедры «Вычислительные системы и сети» Санкт-Петербургского государственного университета аэрокосмического приборостроения

СВЕДЕНИЯ ОБ АВТОРАХ

ЗИКРАТОВ Игорь Алексеевич

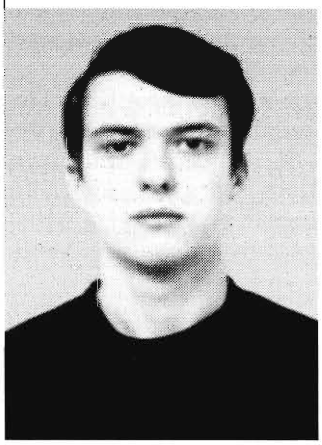


Доцент, начальник кафедры военной кибернетики Санкт-Петербургского высшего военного училища радиоэлектроники.
В 1987 году окончил Киевское высшее инженерное радиотехническое училище ПВО им. маршала авиации А. И. Покрышкина.
В 1999 году защитил диссертацию на соискание ученой степени кандидата технических наук.
Является автором более 40 научных публикаций, в том числе соавтором двух монографий.
Область научных интересов – геоинформационные технологии.

КОЛОТАЕВ Антон Викторович

Ведущий программист фирмы «Транзас технологии», аспирант Санкт-Петербургского института информатики и автоматизации РАН.
В 2002 году окончил математико-механический факультет Санкт-Петербургского государственного университета.
Является автором четырех научных публикаций.
Область научных интересов – обобщенное и производящее программирование, разработка проблемно-ориентированных библиотек.

НАУМОВ Лев Александрович



Аспирант Санкт-Петербургского государственного университета информационных технологий, механики и оптики.
Стипендиат Президента РФ.
Является автором 12 научных публикаций.
Область научных интересов – параллельные вычисления, клеточные автоматы, Grid – системы, вычислительные кластеры.

КОЛЕСОВ Николай Викторович

Профессор, начальник сектора ЦНИИ «Электроприбор».
В 1969 году окончил Ленинградский электротехнический институт.
В 1992 году защитил диссертацию на соискание ученой степени доктора технических наук.
Является автором 120 научных публикаций, в том числе двух монографий.
Область научных интересов – вычислительные системы реального времени, диагностирование вычислительных и информационно-измерительных систем, системы искусственного интеллекта.

МУРОМЦЕВ Дмитрий Юрьевич

Доцент кафедры конструирования радиоэлектронных и микропроцессорных систем Тамбовского государственного технического университета.
В 1996 году окончил Тамбовский государственный технический университет.
В 2000 году защитил диссертацию на соискание ученой степени кандидата технических наук.
Является автором 72 научных публикаций.
Область научных интересов – оптимальное управление динамическими объектами.

ПЕРОВСКАЯ Евгения Ивановна

Профессор кафедры вычислительных систем и сетей Санкт-Петербургского государственного университета аэрокосмического приборостроения. Лауреат Премии Совета министров СССР за 1982 год.
В 1956 году окончила Ленинградский политехнический институт.
В 1982 году защитила диссертацию на соискание ученой степени доктора технических наук.
Является автором более 200 научных публикаций.
Область научных интересов – имитационное моделирование и системы компьютерной поддержки принятия управленческих решений в системе «Человек–Культура–Технологии–Природа».

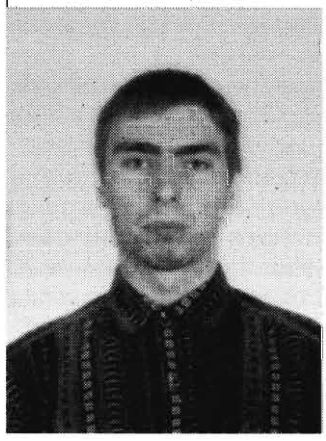
РЫЖИКОВ Юрий Иванович

Профессор кафедры математического обеспечения ЭВМ Военно-космической академии им. А. Ф. Можайского. Заслуженный деятель науки РФ.
В 1958 году окончил Черноморское высшее военно-морское училище.
В 1969 году защитил диссертацию на соискание ученой степени доктора технических наук.
Является автором более 200 научных публикаций, в том числе 40 учебных пособий и монографий.
Область научных интересов – теория очередей, имитационное моделирование, вычислительные методы и прикладное программирование, управление запасами, науковедение и педагогика высшей школы.

ТОЛМАЧЕВА Марина Владимировна

Ведущий инженер ЦНИИ «Электроприбор».
В 1987 году окончила Ленинградский институт точной механики и оптики по специальности «Электронные вычислительные машины».
Является автором 14 научных публикаций.
Область научных интересов – вычислительные системы реального времени, технологии создания и сопровождения программного обеспечения.

САБОНИС Сергей Станиславович



Аспирант кафедры автоматики и вычислительной техники Санкт-Петербургского государственного политехнического университета.
В 2002 году окончил Санкт-Петербургский государственный политехнический университет.
Является автором двух научных публикаций.
Область научных интересов – функциональная диагностика, системный анализ, теория принятия решений, теория автоматического управления, теория оптимизации, теория массового обслуживания.

УДК 638.512.011.56.001.57.681.5

Информационная система энергосберегающего управления сложными объектами

Муромцев Д. Ю. – Информационно-управляющие системы, 2005. – № 5. – С. 2–5.

Рассматривается комплекс задач анализа и синтеза оптимального энергосберегающего управления типовыми динамическими объектами различных классов с учетом возможных изменений состояний функционирования при эксплуатации. Для оперативного решения задач анализа и синтеза оптимального управления используется комбинация методов принципа максимума, динамического программирования и синтезирующих переменных. Приводятся алгоритмы синтеза оптимальных управляющих воздействий в реальном масштабе времени.

Список лит.: 6 назв.

UDK 638.512.011.56.001.57.681.5

Information system of energy-saving complex objects control

Muromtsev D. Yu. –IUS, 2005. N 5. – P. 2–5.

The set of tasks aimed at analysis and synthesis of optimum energy saving control by standard dynamic object of different types with regard for possible changes of functioning states in operation is considered. The combination of methods of maximum principle, dynamic programming and synthesizing variables is applied for operative solution of tasks of analysis and synthesis of optimum control. The algorithms of synthesis of optimum controlled effect in real time scope are given.

Refs: 6 titles.

УДК 681.518.5

Алгоритмы диагностирования автоматизированной системы контроля уровня воды

Сабонис С. С. – Информационно-управляющие системы, 2005. – № 5. – С. 6–10.

В статье рассматривается автоматизированная система контроля уровня воды ОВУ-214 («Водоотливная станция»). Для диагностирования данной системы применяются алгоритмы обнаружения дефектов: модифицированный алгоритм кумулятивных сумм для обнаружения изменения среднего и алгоритм Гиршика–Рубина–Ширяева для обнаружения изменения среднего и дисперсии, в качестве модели диагностирования рассматривается модель аналогового датчика, измеряющего уровень воды в сборной емкости водоотливной станции.

Список лит.: 7 назв.

UDK 681.518.5

Algorithms of diagnosing of water level monitoring automated system

Sabonis S. S. –IUS, 2005. N 5. – P. 6–10.

The water level monitoring automated system is considered in this paper. The modified CUSUM algorithm and the algorithm by Girshik, Rubin and Shiryaev are applied to the system diagnose. The model of the analog sensor, measuring water level, is used as a model of diagnostic object.

Refs: 7 titles.

УДК 681.518

Метод автоматического отождествления линий равных высот при создании цифровых карт местности на основе картометрического подхода

Зикратов И. А. – Информационно-управляющие системы, 2005. – № 5. – С. 11–15.

Предложен метод, основанный на использовании критерия максимального правдоподобия, позволяющий осуществлять отождествление линий равных высот в процессе оцифровки картографического материала при наличии шумов сканирования. Рассмотрены критерии нахождения узловых точек для аппроксимации линий равных высот ортогональными полиномами и построения точек экстраполяции.

Список лит.: 5 назв.

UDK 681.518

Method of automatic identification of equal heights lines during creation of digital region's maps

Zikratov I. A. –IUS, 2005. N 5. – P. 11–15.

The method based on use of maximal plausibility criterion is offered, allowing carrying out identification of equal heights lines during digitization of cartographical material with scanning noises, is offered. Criteria of central points finding for approximation of equal heights lines by orthogonal polynoms and creation of extrapolation points are considered.

Refs: 5 titles.

УДК 65.012.122

Составление расписаний решения задач в конвейерных вычислительных системах

Колесов Н.В., Толмачева М.В. – Информационно-управляющие системы, 2005. – № 5. – С. 16–21.

Рассматриваются алгоритмы составления расписаний в конвейерных вычислительных системах. Анализируются три базовых случая, для которых указываются бесперебороные алгоритмы составления расписаний для случаев, когда времена решения задач точно известны и когда они задаются интервалами.

Список лит.: 5 назв.

UDK 65.012.122

Schedule tabling for tasks solution in pipeline computer systems

Kolesov N. V., Tolmacheva M. V. –IUS, 2005. N 5. – P. 16–21.

The algorithms of Schedule tabling for tasks solution in pipeline computer systems are considered. Three basic cases are analyzed. Author proposes algorithms, which are used for the cases with certain time of task solution specified as intervals.

Refs: 5 titles.

УДК 681.3.06:62-507

Решение задач с помощью клеточных автоматов посредством программного обеспечения CAME&L (Часть I)

Наумов Л.А. – Информационно-управляющие системы, 2005. – № 5. – С. 22–30.

Работа представляет собой введение в программирование для пакета CAME&L посредством библиотеки CADLib. Описываются основы решения задач с помощью рассматриваемого программного обеспечения, а именно – создание пользовательских правил для клеточных автоматов. На примере игры «Жизнь» демонстрируется разработка простейших решений, использование зональной оптимизации, поддержка многопроцессорной и кластерной вычислительных систем, а также – обобщенных координат. Кроме того, приводится пример решения физической задачи, а именно уравнения теплопроводности.

Список лит.: 20 назв.

UDK 681.3.06:62-507

Tasks solution with using bit-mapped automatic devices by means of software CAME&L (Part 1)

Naumov L.A. –IUS, 2005. N 5. – P. 22–30.

This work represents the introduction into programming for CAME&L software by means of CADLib library. Bases of building tasks' solutions with the help of considered software, namely the creation of user rules for cellular automata, are described. Here five variants of "Game of Life" are introduced: plain one, variant with zonal optimization, implementations for multiprocessor and cluster computing systems and for generalized coordinates. Moreover the solution of physical problem, thermal conductivity equation, is described.

Refs: 20 titles.

УДК 519.872; 519.876.5

Расчет многоуровневой системы очередей

Рыжиков Ю.И. – Информационно-управляющие системы, 2005. – № 5. – С. 31–34.

Излагаются соображения в пользу коллективного использования вычислительных ресурсов. Предложен основанный на законе сохранения объема работы метод расчета среднего времени пребывания заявок в одноканальной многоуровневой системе с квантованным обслуживанием в зависимости от заявленного времени счета. Длительности квантов и величина системных потерь предполагаются фиксированными и переменными по уровням.

Список лит.: 7 назв.

UDK 519.872; 519.876.5

Calculation of multilevel system of lines

Ryzhikov Yu. I. –IUS, 2005. N 5. – P. 31–34.

The method of calculation of average time of applications sojourn in single-channel multilevel system with quantum service depending on declared time is offered. Quantum's length and magnitude of system losses are assumed fixed and variables on levels.

Refs: 7 titles.

УДК 621.856

Системный анализ и имитационное моделирование как средство объединения результатов гуманитарных и точных наук

Е. И. Перовская. – Информационно-управляющие системы, 2005. – № 5. – С. 35–46.

Основной задачей является построение математической теории для описания метасистемы объединения разнородных моделей компонент социумов, являющихся результатами исследований как гуманитарных, так и точных наук. Разработанная теория и формальные методы предназначены для междисциплинарных исследований, учитывающих влияние результатов различных аспектов жизнедеятельности общества и природы как на отдельные социумы, так и на цивилизацию в целом при решении проблемы жизнеспособного (устойчивого) развития. Первая часть статьи содержит постановку проблемы создания имитационных моделей систем Человек–Культура–Технологии–Природа.

Список лит.: 6 назв.

УДК 681.3

Реализация библиотеки имитационных моделей как набора обобщенных компонент

А. В. Колотаев. – Информационно-управляющие системы, 2005. – № 5. – С. 47–55.

В статье рассматривается применимость методов обобщенного программирования для реализации библиотеки имитационных моделей на языке C++. Описываемый подход представляется перспективным для разработки моделей больших и сложных систем, поскольку позволяет сочетать эффективность имитационных программ с широкими возможностями повторного использования модулей.

Список лит.: 19 назв.

UDK 621.856

The system analysis and imitating modeling as means of classical and exact sciences results unification.

Perovskaya E. I. –IUS, 2005. N 5. – P. 35–46.

The work is directed on a solution of a fundamental problem of global community models creation for checking of administrative solutions and their consequences without experimenting on Human and Nature. The basic goal is development of the mathematical theory of association of diverse community components models, which are researches results of the both humanitarian and exact sciences in uniform system. The developed theory and formal methods are intended for interdisciplinary researches which take into account influence of results of various aspects of functioning of a society and a nature, both on separate communities, and on a civilization as a whole at the decision of a problem of viable (sustainable) development. The first part of article contains statement of a problem of Human–Culture–Technology–Nature system simulation.

Refs: 6 titles.

UDK 681.3

Implementing of simulation models library with using generalized programming

Kolotaev A. V. –IUS, 2005. N 5. – P. 47–55.

Applicability of generalized programming methods for developing simulation models C++ library is considered. The approach described seems to be valuable for large and complex systems development since it allows creating highly reusable modules without scarifying their efficiency.

Refs: 19 titles.

Фото к статье А. А. Шалыто
"НИКЛАУС ВИРТ – ПОЧЕТНЫЙ ДОКТОР САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ"

Выступление
с докторской лекцией

Вручение профессору Н. Вирту диплома
Почетного доктора СПбГУ ИТМО

Встреча профессора Н. Вирта с руководством СПбГУ ИТМО и чемпионами мира по программированию
(слева направо стоят: проф. А. А.Шалыто, проф. В. Г. Парфенов, А. С. Станкевич, П. Маврин,
А. В. Иванов, Ю. Гутхнехт, проф. В. Л. Макаров;

Общение со студентами