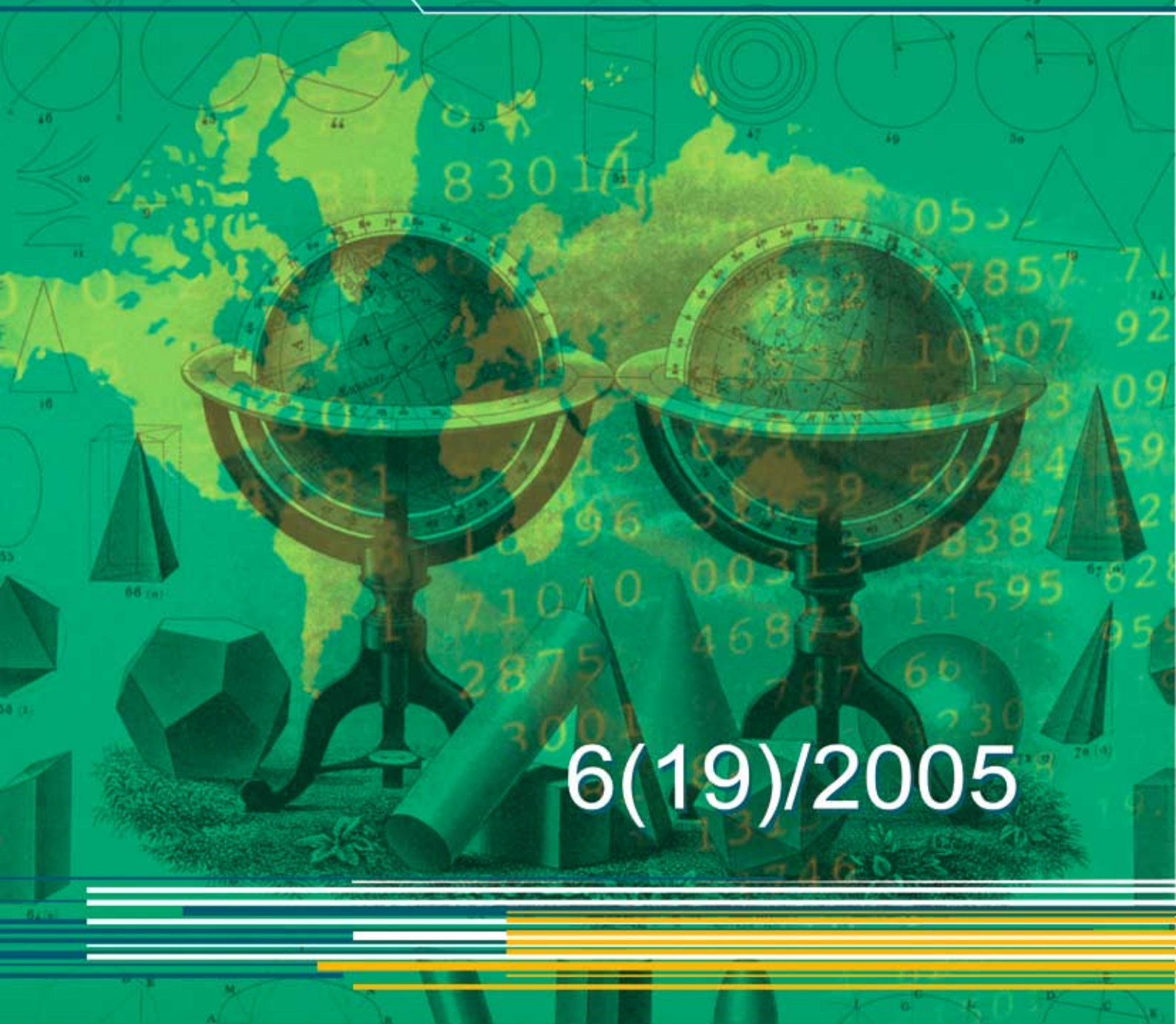
ISSN 1684–8853

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ

НАУЧНО-ПРАКТИЧЕСКИЙ ЖУРНАЛ



6(19)/2005

НАУЧНО-ПРАКТИЧЕСКИЙ ЖУРНАЛ

6(19)/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.
«Роспечать» («Газеты и журналы») – № 15385.

ЛР № 010292 от 18.08.98.
Сдано в набор 30.10.2005. Подписано в печать 02.12.2005. Формат 60×90 1/8.
Бумага офсетная. Гарнитура SchoolBookC. Печать офсетная.
Усл. печ. л. 8,0. Уч.-изд. л. 9,0. Тираж 1000 экз. Заказ 563.

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 621.391.826; 681.5

МАТЕМАТИЧЕСКИЕ МОДЕЛИ ЭХО-СИГНАЛОВ КОРАБЛЕЙ, НАБЛЮДАЕМЫХ ЛОКАТОРАМИ БОРТОВЫХ СИСТЕМ ОБРАБОТКИ ИНФОРМАЦИИ

А. Г. Давидчук,
ассистент
Д. А. Шепета,
канд. техн. наук, доцент
Санкт-Петербургский государственный университет аэрокосмического приборостроения

Предлагается математическая модель эхо-сигналов кораблей, построенная на основе экспериментальных данных, которая позволяет учитывать флюктуации амплитуд и длительностей сигналов, а также их взаимные корреляционно-спектральные характеристики. Математическая модель основана на логарифмически нормальном многомерном законе распределения флуктуаций амплитуд и длительностей.

The mathematical model of ships' echo – signals, constructed on the experimental data basis which allows to take into account fluctuations of amplitudes, durations of signals, and mutual correlation and spectral characteristics is offered. The mathematical model is based on the logarithmic-normal multidimensional law of distribution fluctuations of amplitudes and duration.

При построении математических моделей входных сигналов бортовых систем обработки информации применяются в основном три подхода: первый состоит в постулировании математической модели; второй основан на изучении физических явлений, обуславливающих появление сигналов; третий использует «статистический эквивалент» сигналов, построенный по экспериментальным данным. Под математической моделью в данной статье понимается статистическая модель – многомерный совместный закон распределения параметров сигналов, наблюдаемых на выходе приемного устройства бортового локатора. За основу принят третий подход, который при построении модели учитывает выводы относительно статистических характеристик наблюдаемых сигналов (приведенные в литературе по обработке экспериментальных данных).

Математическая модель информационного сигнала на входе локационного тракта

При поиске, обнаружении, идентификации, выборе, захвате на автосопровождение и сопровождение корабля математическая модель информационного сигнала зависит, по крайней мере, от пяти основных факторов: характеристик самого корабля, тракта распространения электромагнитных волн, характеристик зондирующего сигнала, условий наблюдения корабля, приемного тракта бортового локатора [1, 4, 9–12]. Определяющими факторами являются характеристики корабля и вид излучаемого сигнала [9, 12]. Остальные факторы можно учесть следующим образом: характеристики тракта – изменением параметров модели, условия наблюдения – введением функциональных зависимостей между условиями наблюдения и параметрами модели. Тракт обработки считаем линейным и широкополосным, что позволяет рассматривать независимо модели информационного сигнала и помех.

Под информационным сигналом, в соответствии с принятыми допущениями, будем понимать отраженный физическим объектом зондирующий сигнал, прошедший тракт распространения и приемное устройство бортового локатора. После гетеродина приемного устройства наблюдаемый ин-

формационный сигнал $\tilde{S}(t)$ с точностью до коэффициента пропорциональности можно записать в виде

$$\tilde{S}(t)=\sum_{i=1}^{N}\sum_{j=1}^{M}k_{0,j}A_{i,j}S\left(t-t_{0,j}-2\tau_c\left(N-1\right)\right)\times$$
$$\times\cos\left(\omega_0 t+\varphi_{c,i,j}+\varphi_{0,j}\right), \qquad (1)$$

где $t_{0,j}$ – задержка j-й пачки сигнала; $S(t)$ – нормированная огибающая импульсов; ω_0 – промежуточная частота последнего каскада приемного устройства; M – число импульсов в наблюдаемой пачке; N – число принимаемых пачек импульсов.

Здесь принято, что за время отражения пачки условия отражения не меняются, что всегда выполняется в реальных условиях наблюдения морских целей (при импульсно-пачечном режиме). Выражение (1) представляет собой достаточно общую форму записи сигнала, позволяющую учесть фазовую, частотную, амплитудную и временную (период следования пачек импульсов) модуляцию (манипуляцию) сигнала. Частными случаями этого выражения являются выражения для одиночного импульсного сигнала, ЛЧМ-сигнала (при соответствующем предельном переходе), ФМ-сигнала, сигнала со случайной или детерминированной вобуляцией периода повторения.

После амплитудно-фазовой обработки сигнала j-я пачка полностью определяется характеристиками лишь одного своего импульса, а именно: его амплитудой $A_j = k_{0,j}A_3$, начальной фазой $\varphi_j = \varphi_{0,j}$, задержкой пачки $t_{0,j}$. Задержку пачки $t_{0,j}$ будем считать детерминированным параметром сигнала, что допустимо, если пренебречь дальномерным шумом. Тогда каждая пачка характеризуется случайными величинами A_j, φ_j и огибающей $S_j(t)$.

Таким образом, при введенных допущениях информационный сигнал можно представить в виде последовательности импульсных сигналов; i-й импульс этого сигнала запишем в виде $S_i(t) = S_i(t)\cos(\omega_0 t+\varphi_i)$, где $S_i(t)$ – огибающая импульса; ω_0 – частота (может быть равной нулю); φ_i – начальная фаза (измеренная относительно сигнала, начальная фаза которого условно принята за нуль).

Данная последовательность импульсов (пачка импульсов) может быть когерентной (начальные фазы импульсов связаны друг с другом известным соотношением) и некогерентной. Рассмотрим случай некогерентной пачки импульсов, так как при наших допущениях это более общий случай (в том смысле, что для когерентной пачки достаточно задать начальную фазу лишь одного из импульсов), при этом частоту можно считать известной детерминированной величиной. Для некогерентной последовательности начальные фазы φ_i, $i = 1, 2, \ldots, N$, представляют собой совместно независимые случайные величины, распределенные равномерно на интервале $[0, 2\pi)$. Огибающая $S_i(t)$ представляет собой известную функцию, зависящую от двух параметров: амплитуды сигнала $A_i = \max S_i(t)$ и его длительности τ_i, определенной по уровню 0,5, т. е. $\tau_i = |t_i'' - t_i'|$, где t_i'' и t_i' – два крайних члена из упорядоченного ряда решения уравнения $S_i(t) = 0{,}5A_i$.

В качестве S_i обычно используют следующие функции:

– прямоугольную

$$S_i(t)=\begin{cases}0, & t<-0{,}5\tau_i,\\ A_i, & |t|\le 0{,}5\tau_i,\\ 0, & t>0{,}5\tau_i;\end{cases} \qquad (2)$$

– треугольную

$$S_i(t)=\begin{cases}0, & \tau_i<|t|,\\ A_i+\tau_i^{-1}A_i t, & -\tau_i\le t<0,\\ A_i-\tau_i^{-1}A_i t, & 0\le t\le\tau_i;\end{cases} \qquad (3)$$

– гауссову

$$S_i(t)=A_i\exp\{-4\ln 2\,\tau_i^{-2}t^2\}. \qquad (4)$$

Если длительность зондирующих импульсов сравнима с радиолокационным размером корабля, то форма эхо-сигналов близка к гауссовской кривой [2, 5, 9].

Таким образом, наблюдаемый импульс характеризуется четырьмя параметрами: амплитудой A_i, длительностью τ_i, начальной фазой φ_i, частотой ω_0. Три первых параметра случайные, а четвертый – $\omega_0 = \text{const}$. Следовательно, полезный сигнал полностью определяется случайной трехмерной величиной $\gamma_i = \{A_i, \tau_i, \varphi_i\}$ и параметром ω_0. Поэтому для формирования математической модели сигнала необходимо задать параметр ω_0 и многомерный закон (функцию или плотность распределения) случайных величин $\gamma_1, \gamma_2, \ldots, \gamma_N$.

Для рассматриваемых некогерентных сигналов случайные многомерные величины $\{A_1, \tau_1, \ldots, A_N, \tau_N\}$ и $\{\varphi_1, \ldots, \varphi_N\}$ взаимно независимы, а φ_i – равномерно распределены на интервале $[0, 2\pi)$. При любых i и N многомерная плотность распределения равна

$$f(\gamma_1,\ldots,\gamma_{iN})=f(A_1,\tau_1,\ldots,A_N,\tau_N)\prod_{j=1}^{N}f(\varphi_j), \qquad (5)$$

где

$$f(\varphi_i)=\begin{cases}0, & \varphi_j<0,\\ (2\pi)^{-1}, & 0\le\varphi_i<2\pi,\\ 0, & \varphi_j\ge 2\pi.\end{cases} \qquad (6)$$

Для когерентной пачки выражение (6) будет содержать произведение из $N-1$ дельта-функций.

Из выражения (6) следует, что для получения математической модели сигнала достаточно задать многомерную плотность распределения $f(A_1, \tau_1, A_2, \tau_2, \ldots, A_N, \tau_N)$, так как статистические характеристики флюктуаций фаз определены выражением (6). Следовательно, для математического описания полезного сигнала необходимо определить плотность распределения $f(A_1, \tau_1, A_2, \tau_2, \ldots, A_N, \tau_N) = f(\mathbf{A}_N, \tau_N)$, $N = (1, 2, \ldots)$, где $\mathbf{A}_N = \{A_1, A_2, \ldots, A_N\}$, $\tau_N = \{\tau_1, \tau_2, \ldots, \tau_N\}$ ($\mathbf{A}_N$ и τ_N – векторы амплитуд и длительностей сигналов соответственно).

Плотность распределения амплитуд и длительностей локационных сигналов, отраженных от надводных объектов

В качестве одномерных функций распределения амплитуд использовались разные функции распределения: Рэлея, Рэлея–Райса, Накагами, логарифмически нормальная, хи-квадрат и др. [3, 6–8, 11, 12]. Наиболее часто в последнее время используется логарифмически нормальная плотность распределения амплитуд, так как она очень хорошо согласуется с экспериментальными данными, полученными для различных типов кораблей и различных условий их наблюдения. Эту плотность распределения и будем использовать для построения многомерной модели эхо-сигнала корабля.

Одномерная функция плотности распределения амплитуды A_i локационного импульса, распределенного по логарифмически нормальному закону, определяется по формуле

$$f(A_i) = \frac{1}{\sqrt{2\pi}\sigma_{A_i} A_i} \exp\left\{-\frac{\ln^2 \frac{A_i}{\bar{A}_i}}{2\sigma^2_{A_i}}\right\}, \qquad (7)$$

где $\bar{A}_i$ и σ_i – параметры распределения, причем $\bar{A}_i$ и σ_i связаны со средним $\tilde{A}_i$ и дисперсией $\tilde{D}_{A_i}$ распределения следующими соотношениями:

$$\begin{cases} \tilde{A}_i = \bar{A}_i \exp\left\{\dfrac{\sigma^2_{A_i}}{2}\right\}, \\ \tilde{D}_{A_i} = \tilde{A}_i^2 \left[\left(\dfrac{\tilde{A}_i}{\bar{A}_i}\right)^2 - 1\right]. \end{cases} \qquad (8)$$

Однако на практике вместо $\tilde{D}_{A_i}$ для характеристик флюктуаций амплитуд локационного сигнала используют коэффициент вариации $K_{A_i} = \dfrac{\sqrt{\tilde{D}_{A_i}}}{\tilde{A}_i}$.

В этом случае из выражения (8) получим

$$\bar{A}_i = \frac{\tilde{A}_i}{\sqrt{1+K^2_{A_i}}}; \quad \sigma_{A_i} = \sqrt{\ln(1+K^2_{A_i})}. \qquad (9)$$

Иногда вместо коэффициента вариации по амплитуде K_{A_i} используют коэффициент вариации по мощности K_{P_i}. Плотность распределения мощности f_{P_i} также логарифмически нормальна с параметрами:

$$\bar{P}_i = \frac{\tilde{P}_i}{\sqrt{1+K^2_{P_i}}} = \frac{\bar{A}_i^2}{2}, \quad \sigma_{P_i} = \sqrt{\ln(1+K^2_{A_i})} = 2\sigma_{A_i}, \qquad (10)$$

где $\tilde{P}_i$ – среднее значение мощности; $1+K^2_{P_i} = (1+K^2_{A_i})^4$; значение $\tilde{P}_i$ определяется с учетом параметров локационной структуры по известной формуле

$$\tilde{P}_i = \frac{P_{\text{пер}_i} G_i^2 \eta_i \lambda_i^2 \tilde{S}_i}{(4\pi)^3 R_i^4}, \qquad (11)$$

где $P_{\text{пер}_i}$ – мощность передатчика при излучении i-го импульса; G_i^2 – коэффициент усиления антенны по мощности (при излучении и приеме на одну антенну); λ_i – длина волны передатчика; η_i – коэффициент потерь, $\eta_i \approx 0{,}25$; $\tilde{S}_i$ – средняя эффективная поверхность рассеивания (ЭПР) при i-м зондировании; R_i – расстояние между антенной и объектом.

Выражения (9)–(11) полностью определяют параметры одномерной плотности распределения амплитуд локационных сигналов.

Для аппроксимации одномерной плотности распределения длительностей импульсов эхо-сигналов кораблей предлагались различные законы распределения: хи-квадрат, нормальный, логарифмически нормальный, усеченный нормальный. Наиболее простыми являются нормальный и логарифмически нормальный. Следует отметить, что использование нормального закона некорректно в силу ненулевой вероятности появления отрицательных значений длительности. Переход от нормального закона к усеченному нормальному приводит к корректным выражениям, но настолько усложняет модель, что делает ее практически бесполезной. Все перечисленные законы (с учетом оговорки для нормального закона) достаточно хорошо согласуются с экспериментальными характеристиками, и поэтому при выборе закона (из перечисленных) следует исходить из соображений простоты модели, ее удобства для аналитических вычислений и синтеза алгоритмов моделирования.

В силу сказанного выше имеет смысл принять статистические характеристики флюктуаций τ_i такие же, как и для амплитуд. Физически флюктуации длительности отраженных импульсов определяются теми же причинами, что и флюктуации амплитуд.

Итак, примем, что длительность отраженного импульса распределена по логарифмически нормальному закону. Тогда функция плотности распределения определяется следующим выражением:

$$f(\tau_i)=\frac{1}{\sqrt{2\pi}\sigma_{\tau_i}\tau_i}\exp\left\{-\frac{\ln^2\frac{\tilde{\tau}_i}{\overline{\tau}_i}}{2\sigma^2_{\tau_i}}\right\}, \qquad (12)$$

где параметры распределения так же, как и в случае с амплитудой, равны

(13)

(и – средняя длительность и коэффициент вариации длительности i-го импульса соответственно).

Таким образом, одномерные плотности распределения амплитуд и длительностей эхо-сигналов кораблей полностью определены. Эти плотности распределения хорошо согласуются с экспериментальными данными, имеют достаточно простые аналитические выражения, которые удобно использовать в аналитических выкладках и при синтезе практических алгоритмов моделирования флюктуаций амплитуд и длительностей эхо-сигналов кораблей.

Двумерные плотности распределения и корреляционные функции амплитуд и длительностей

Знания одномерных функций распределения амплитуды и длительности отраженных сигналов еще недостаточно для записи двумерной функции плотности распределения этих величин. Для определения двумерных плотностей распределения амплитуд, длительностей, амплитуд и длительностей необходим очень большой объем выборочных данных, к тому же полученных при одних и тех же условиях, что практически невозможно. Поэтому на практике оценивают лишь одномерные законы распределения амплитуд и длительностей, а также ковариационные функции амплитуд, длительностей и взаимную корреляционную функцию амплитуд и длительностей. Затем подбирают такие двумерные плотности распределения, для которых маргинальные плотности распределения и ковариационные зависимости не противоречат экспериментальным данным. Эти двумерные плотности распределения и постулируются в качестве математических моделей, описывающих совместные флюктуации амплитуд и длительностей. Используем эту методику для получения соответствующих двумерных плотностей распределения.

На двумерные плотности распределения накладываем следующие требования: маргинальные плотности должны совпадать с оценками одномерных плотностей для амплитуд, длительностей, амплитуд и длительностей; коэффициенты корреляции амплитуд, длительностей, амплитуд и длительностей должны совпадать с соответствующими оценками коэффициентов корреляции. Этим требованиям удовлетворяют двумерные логарифмически нормальные плотности распределения.

Для амплитуд двумерная логарифмически нормальная плотность распределения записывается в виде

$$f(A_i,A_j)=\frac{1}{2\pi\sigma_{A_i}\sigma_{A_j}A_iA_j\sqrt{1-r^2_{A_{ij}}}}\exp\left\{-\frac{1}{2(1-r^2_{A_{ij}})}\times\right.$$

$$\left.\times\left[\frac{\ln^2\frac{A_i}{\overline{A}_i}}{\sigma^2_{A_i}}+\frac{\ln^2\frac{A_j}{\overline{A}_j}}{\sigma^2_{A_j}}-2r_{A_{ij}}\frac{\ln\frac{A_i}{\overline{A}_i}}{\sigma_{A_i}}\frac{\ln\frac{A_j}{\overline{A}_j}}{\sigma_{A_j}}\right]\right\}, \qquad (14)$$

где – параметр распределения.

Коэффициент корреляции амплитуд $R_{A_{ij}}$ определяется следующим образом:

$$R_{A_{ij}}=\frac{1}{\sqrt{\tilde{D}_{A_i}\tilde{D}_{A_j}}}\int_0^\infty\int_0^\infty(A_i-\tilde{A}_j)(A_j-\tilde{A}_i)f(A_i,A_j)dA_idA_j=$$

$$\frac{K_{A_{ij}}}{\tau_i}=\frac{\tilde{\tau}\sqrt{\tilde{D}_{A_i}\tilde{D}_{A_j}}}{\sqrt{1+K^2_{\tau_i}}},\quad \sigma_{\tau_i}=\sqrt{\ln(1+K^2_{\tau_i})},$$

$$=\frac{1}{\sqrt{\tilde{D}_{A_i}\tilde{D}_{A_j}}}\left[\int_0^\infty\int_0^\infty A_iA_jf(A_i,A_j)dA_idA_j-\tilde{A}_i\tilde{A}_j\right]=$$

$$=\frac{\tilde{A}_i\tilde{A}_j}{\sqrt{\tilde{D}_{A_i}\tilde{D}_{A_j}}}\left(\mathbf{e}^{r_{A_{ij}}\sigma_{A_i}\sigma_{A_{(j-1)}}}\right)=\frac{\mathbf{e}^{r_{A_{ij}}\sigma_{A_i}\sigma_{A_j}}}{K_{A_i}K_{A_j}}. \qquad (15)$$

Из этого выражения можно найти $r_{A_{ij}}$ в виде

$$r_{A_{ij}}=\frac{\ln(1+K_{A_i}K_{A_j}R_{A_{ij}})}{\sqrt{\ln(1+K^2_{A_i})\ln(1+K^2_{A_j})}}. \qquad (16)$$

Двумерную плотность распределения длительностей импульсов также положим логарифмически нормальной:

$$f(\tau_i,\tau_j)=\frac{1}{2\pi\sigma_{\tau_i}\sigma_{\tau_j}\tau_i\tau_j\sqrt{1-r^2_{\tau_{ij}}}}\exp\left\{-\frac{1}{2(1-r^2_{\tau_{ij}})}\times\right.$$

$$\left.\times\left[\frac{\ln^2\frac{\tau_i}{\overline{\tau}_i}}{\sigma^2_{\tau_i}}+\frac{\ln^2\frac{\tau_j}{\overline{\tau}_j}}{\sigma^2_{\tau_j}}-2r_{\tau_{ij}}\frac{\ln\frac{\tau_i}{\overline{\tau}_i}}{\sigma_{\tau_i}}\frac{\ln\frac{\tau_j}{\overline{\tau}_j}}{\sigma_{\tau_j}}\right]\right\}, \qquad (17)$$

где $r_{\tau_{ij}}$ – параметр распределения, определяемый, по аналогии с амплитудой, через коэффициент корреляции длительностей:

$$r_{\tau_{ij}} = \frac{\ln(1+K_{\tau_i}K_{\tau_j}R_{\tau_{ij}})}{\sqrt{\ln(1+K_{\tau_i}^2)\ln(1+K_{\tau_j}^2)}}. \qquad (18)$$

Двумерную совместную плотность распределения амплитуд и длительностей также будем считать логарифмически нормальной:

$$f(\tau_i, A_j) = \frac{1}{2\pi\sigma_{\tau_i}\sigma_{A_j}\tau_i A_j\sqrt{1-r_{\tau_i A_j}^2}}\exp\left\{-\frac{1}{2(1-r_{\tau_i A_j}^2)}\times\right.$$

$$\left.\times\left[\frac{\ln^2\frac{\tau_i}{\overline{\tau}_i}}{\sigma_{\tau_i}^2}+\frac{\ln^2\frac{A_j}{\overline{A}_j}}{\sigma_{A_j}^2}-2r_{\tau_i A_j}\frac{\ln\frac{\tau_i}{\overline{\tau}_i}\ln\frac{A_j}{\overline{A}_j}}{\sigma_{\tau_i}\sigma_{A_j}}\right]\right\}, \qquad (19)$$

где $r_{\tau_i A_j}$ – параметр распределения, определяемый через коэффициент корреляции амплитуд и длительностей $R_{\tau_i A_j}$:

$$r_{\tau_i A_j} = \frac{\ln(1+K_{\tau_i}K_{A_j}R_{\tau_i A_j})}{\sqrt{\ln(1+K_{\tau_i}^2)\ln(1+K_{A_j}^2)}}. \qquad (20)$$

Зная корреляционные функции локационных сигналов, можно определить численные значения $R_{A_{ji}}$, $R_{\tau_{ij}}$ и $R_{\tau_i A_j}$ для различных конкретных условий работы.

Таким образом, двумерные плотности распределения амплитуд, длительностей, амплитуд и длительностей полностью определены. Во многих практических работах на этом построение модели заканчивается, так как двумерных плотностей распределения вполне достаточно, чтобы определять корреляционные характеристики входных процессов систем обработки информации после любого линейного и нелинейного преобразования. Однако в данной работе построим многомерную совместную плотность распределения амплитуд и длительностей, которую можно использовать не только для приближенного анализа систем обработки информации, но в рамках построенной модели и для точного определения характеристик систем или, при приближенном оценивании характеристик, для определения погрешности соответствующих оценок.

Многомерная математическая модель флюктуаций амплитуд и длительностей

Для построения многомерной функции плотности распределения амплитуд и многомерной функции плотности распределения длительностей принимаемого сигнала необходимо учесть следующие основные обстоятельства: двумерные плотности распределения должны быть логарифмически нормальные; значение коэффициентов корреляции амплитуд, длительностей, амплитуд и длительностей должны быть равны заданным.

Для амплитуд и длительностей этим условиям удовлетворяют следующие функции [2,9,13,14]:

$$f_N(\mathbf{A}_N) = \frac{1}{\sqrt{(2\pi)^N D_N^{(A)}}\prod_{i=1}^{N}A_i\sigma_{A_i}}\times$$

$$\times\exp\left\{-\frac{1}{2D_N^{(A)}}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{D_{ij}^{(A)}}{\sigma_{A_i}\sigma_{A_j}}\ln\frac{A_i}{\overline{A}_i}\ln\frac{A_j}{\overline{A}_j}\right\}; \qquad (21)$$

$$f_N(\boldsymbol{\tau}_N) = \frac{1}{\sqrt{(2\pi)^N D_N^{(\tau)}}\prod_{i=1}^{N}\tau_i\sigma_{\tau_i}}\times$$

$$\times\exp\left\{-\frac{1}{2D_N^{(\tau)}}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{D_{ij}^{(\tau)}}{\sigma_{\tau_i}\sigma_{\tau_j}}\ln\frac{\tau_i}{\overline{\tau}_i}\ln\frac{\tau_j}{\overline{\tau}_j}\right\}, \qquad (22)$$

где определители соответственно равны:

$$D_N^{(A)} = \begin{vmatrix} 1 & r_{A_{12}} & \dots & r_{A_{1N}} \\ r_{A_{21}} & 1 & \dots & r_{A_{2N}} \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ r_{A_{N1}} & r_{A_{N2}} & \dots & 1 \end{vmatrix}; \qquad (23)$$

$$D_N^{(\tau)} = \begin{vmatrix} 1 & r_{\tau_{12}} & \dots & r_{\tau_{1N}} \\ r_{\tau_{21}} & 1 & \dots & r_{\tau_{2N}} \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ r_{\tau_{N1}} & r_{\tau_{N2}} & \dots & 1 \end{vmatrix}. \qquad (24)$$

В этих формулах $D_{ij}^{(A)}$ и $D_{ij}^{(\tau)}$ – алгебраические дополнения элементов $r_{A_{ij}}$ и $r_{\tau_{ij}}$ в определителях $D_N^{(A)}$ и $D_N^{(\tau)}$ соответственно. При этом $r_{A_{ij}}$ и $r_{\tau_{ij}}$ определяются по формулам (16) и (18).

Для построения многомерной функции плотности распределения амплитуд и длительностей сигнала уже необходимо потребовать выполне-

ния трех условий: многомерная функция плотности распределения амплитуд должна иметь вид (21); многомерная функция плотности распределения длительностей должна иметь вид (22); взаимная корреляционная функция амплитуд и длительностей должна соответствовать заданной.

Всем этим трем условиям удовлетворяет функция

$$f_{2N}(\mathbf{A}_N,\tau_N)=\frac{1}{(2\pi)^N\sqrt{D_{2N}}\prod_{i=1}^{N}A_i\sigma_{A_i}\tau_i\sigma_{\tau_i}}\times$$

$$\times\exp\left\{-\frac{1}{2D_{2N}}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{D_{ij}}{\sigma_{A_i}\sigma_{A_j}}\ln\frac{A_i}{\overline{A}_i}\ln\frac{A_j}{\overline{A}_j}+\right.$$

$$+\frac{D_{i,N+j}}{\sigma_{A_i}\sigma_{\tau_j}}\ln\frac{\tau_j}{\overline{\tau}_j}+\frac{D_{N+i,j}}{\sigma_{\tau_i}\sigma_{A_j}}\ln\frac{\tau_i}{\overline{\tau}_i}\ln\frac{A_j}{\overline{A}_j}+$$

$$\left.+\frac{D_{N+i,N+j}}{\sigma_{\tau_i}\sigma_{\tau_j}}\ln\frac{\tau_i}{\overline{\tau}_i}\ln\frac{\tau_j}{\overline{\tau}_j}\ln\frac{\tau_i}{\overline{\tau}_i}\right\}, \qquad (25)$$

у которой определитель вычисляется следующим образом:

$$D_{2N}=\begin{vmatrix}
1 & r_{A_{12}} & \dots & r_{A_{1N}} & r_{A_1\tau_1} & r_{A_1\tau_2} & \dots & r_{A_1\tau_N}\\
r_{A_{21}} & 1 & \dots & r_{A_{2N}} & r_{A_2\tau_1} & r_{A_2\tau_2} & \dots & r_{A_2\tau_N}\\
\cdot & \cdot & & & \cdot & \cdot & & \\
\cdot & \cdot & & & \cdot & \cdot & & \\
\cdot & \cdot & & & \cdot & \cdot & & \\
r_{A_{N1}} & & \dots & 1 & r_{A_N\tau_1} & & \dots & r_{A_N\tau_N}\\
r_{\tau_1A_1} & r_{\tau_1A_2} & \dots & r_{\tau_1A_N} & 1 & r_{\tau_{12}} & \dots & r_{\tau_{1N}}\\
r_{\tau_2A_1} & r_{\tau_2A_2} & \dots & r_{\tau_2A_N} & & r_{\tau_{21}} & \dots & r_{\tau_{2N}}\\
\cdot & \cdot & & & \cdot & \cdot & & \\
\cdot & \cdot & & & \cdot & \cdot & & \\
\cdot & \cdot & & & \cdot & \cdot & & \\
r_{\tau_NA_1} & & \dots & r_{\tau_NA_N} & & r_{\tau_{N1}} & \dots & 1
\end{vmatrix}, \qquad (26)$$

а $D_{l,M}$ – алгебраические дополнения соответствующих элементов определителя D_{2N}. При этом элементы $r_{A_i\tau_j}$ и $r_{\tau_iA_j}$ определяются по формуле (20).

В качестве многомерной плотности распределения $f(\mathbf{A}_N, \tau_N)$ и будем использовать логарифмически нормальную плотность, определяемую выражениями (25) и (26), так как она удовлетворяет перечисленным выше требованиям, а ее частные случаи (маргинальные плотности распределения) не противоречат экспериментальным данным.

Итак, получена функция плотности распределения амплитуд и длительностей, локационного сигнала, отраженного от объекта, имеющая следующие свойства: маргинальные одномерные функции распределения амплитуд и длительностей сигналов имеют логарифмически нормальное распределение; корреляционная функция амплитуд, корреляционная функция длительностей и взаимная корреляционная функция амплитуд и длительностей могут иметь произвольный функциональный вид, при котором матрицы $\left|D_N^{(A)}\right|$, $\left|D_N^{(\tau)}\right|$ и $\left|D_{2N}\right|$ положительно определены (данные функции могут быть заданы и в виде таблиц экспериментальных данных).

Выводы

Математическая модель информационного сигнала представляет собой статистическую модель флюктуаций параметров эхо-сигнала корабля. Эта модель зависит от пяти основных факторов: характеристик объекта, тракта распространения электромагнитных волн, характеристик зондирующего сигнала, условий наблюдения объекта, приемного тракта бортового локатора. Определяющими факторами являются характеристики объекта и вид излучаемого сигнала.

Наиболее распространенными законами распределения параметров модели (амплитуды сигнала) являются: закон Рэлея, Рэлея–Райса, хи-квадрат, закон Накагами, логарифмически нормальный закон. Наиболее целесообразным является использование логарифмически нормальной модели информационного сигнала. В качестве модели флюктуаций длительности отраженных импульсов целесообразно также использовать логарифмически нормальный закон, так как он не только не противоречит экспериментальным данным, но и удобен при построении многомерных моделей.

В качестве многомерного распределения амплитуд и длительностей информационного сигнала используется многомерное логарифмически нормальное распределение, на котором и основана статистическая многомерная математическая модель эхо-сигналов корабля. Построенная модель позволяет учитывать корреляционно-спектральные характеристики амплитуд и длительностей сигналов, а также совместные корреляционно-спектральные характеристики амплитуд и длительностей.

Литература

1. **Андросов В. А., Енатко И. В.** Задачи и принципы построения стендово-имитационной среды для отработки интегрированных комплексов бортового оборудования // Радиотехника. 1996. Вып. 17. С. 120–123.
2. **Бестугин А. Р., Шепета А. П.** Синтез оптимальных алгоритмов классификации и выбора морских объектов в дальней зоне наблюдения // Сб. науч. трудов «Информационно-управляющие системы и сети. Структуры, моделирование, алгоритмы». СПб.: Политехника, 1999. С. 142–152.
3. **Изранцев В. В., Шепета Д. А.** Моделирование внешних сигналов бортовых приборных комплексов летательных аппаратов пятого поколения // Изв. вузов. Сер. Приборостроение. 2000. № 2. С.76–83.
4. **Исаев С. А., Кондратенков Г. С.** Цифро-натурные и летно-модельные методы испытаний КБО // Радиотехника. 1996. Вып. 17. С. 97–100.
5. **Култышев Е. И., Лемешко Н. А., Шепета А. П.** Оценка средней амплитуды и средней длительности локационных импульсов, отраженных от надводного объекта// Тр. Междунар. научно-технич. конф. (Тез. докл.), май 1994. Киев: АН Украины; НПО Квант. Вып. 2. С. 58.
6. **Лемешко Н. А. Шепета Д. А.** Предельные вероятности распознавания медленно флюктуирующих объектов по средней эффективной отражающей поверхности// Тр./ СПбГААП. СПб., 1995. С. 56–60.
7. Обнаружение радиосигналов / П. С. Акимов, Ф. Ф. Евстратов, И. С. Захаров и др.; Под ред. А. А. Колосова. М.: Радио и связь, 1989. 288 с.
8. О вероятности выбора главной цели / А. А. Оводенко, А. П. Шепета и др.: Межвуз. сб. / ЛЭТИ. Л., 1977. Вып. 118. С. 122–124.
9. **Оводенко А. А., Култышев Е. И., Шепета А. П.** Бортовая радиоэлектронная аппаратура/ МПИ. М., 1989. 335 с.
10. Оценка эффективной отражающей поверхности надводной цели по коррелированной выборке / А. В. Геллер, А. П. Шепета и др.// Тр./ ЛИАП. Л., 1974. Вып. 88. С. 96–98.
11. Селекция и распознавание на основе локационной информации/А. Л. Горелик, Ю. Л. Барабаш, О. В. Кривошеев, С. С. Эпштейн; Под ред. А. Л. Горелика. М.: Радио и связь, 1990. 240 с.
12. **Тверской Г. Н., Терентьев Г. К., Харченко И. П.** Имитаторы эхо-сигналов судовых радиолокационных станций. Л.: Судостроение, 1973. 228 с.
13. **Шепета Д. А.** Моделирование входных сигналов бортовых информационно-измерительных систем. Ассоциация инженерных вузов. Академия космонавтики. МГАТУ им. К. Э. Циолковского// XX Гагаринские чтения: Тез. докл. молодежной научн.-технич. конф., апрель 1994. М.: МГАТУ, 1994. Ч. 4. 67 с.
14. **Izrantsev V. V., Kosenkov A. M., Shepeta D. A.** Algorithms For Generating The Non-Gaussian Series With The Given Correlation Characteristics// International Symposium On Problems Of Modular Information Computer Systems And Networks. Abstracts. Moscow; – St.-Petersburg: IEEE, International Informatization Academy, Russian Academy Of Science, Moscow State University, 1997. P. 58.

УДК 602-507

ВИЗУАЛЬНОЕ КОНСТРУИРОВАНИЕ ПРОГРАММ

Ф. А. Новиков,
канд. физ.-мат. наук, доцент
Санкт-Петербургский государственный политехнический университет

В статье описывается подход к проектированию и реализации набора компонентов, которые в совокупности составляют инструментальное средство, основанное на языке UML и поддерживающее все фазы процесса разработки приложений.

An approach to designing and implementing a set of components that form an instrumental tool, based on UML and supporting all phases of application development, is described in the article.

Введение

В настоящее время эффективность и результативность процесса разработки программного обеспечения в целом оставляет желать лучшего. Несмотря на заметные успехи технологии программирования, остается весьма значительной доля проектов по разработке программного обеспечения, которые нельзя считать вполне успешными. Наряду с эффектными достижениями имеются и сравнительно многочисленные досадные неудачи (обзор современного состояния см., например, в работе [1]). К сожалению, до сих пор слишком часто приходится делать вывод, что программирование – это рискованный бизнес, программы ненадежны, а программисты неуправляемы.

Дальнейший прогресс в области совершенствования процесса разработки зависит от множества факторов, таких как совершенствование аппаратной базы компьютеров, внедрение современных форм организации коллективного труда разработчиков, разработка и использование новых информационных технологий и т. д.

Многие полагают, и автор разделяет это мнение, что одним из ключевых факторов является совершенствование методов и инструментов визуального моделирования, а также (в перспективе) конструирования законченных приложений. Ожидается, что языки визуального моделирования должны стать такой же абстракцией над языками программирования, какой языки программирования являются над аппаратной платформой. Именно эта идея положена в основу инициативы, названной MDA (Model Driven Architecture) – архитектура, управляемая моделью [2]. Разработку на основе моделей будем называть модельно-ориентированной разработкой.

Наиболее многообещающими в этой области за последние годы представляются появление и распространение унифицированного языка моделирования UML [3]. Разумеется, UML – это исторически далеко не первая попытка ввести в программирование чертежи, понятные всем. Введение же в повсеместную практику программирования чертежей (как определяющих документов, а не просто иллюстраций) – признак того, что программирование действительно является инженерной областью деятельности. Пока же, по мнению автора, термин «инженер-программист» является скорее благим пожеланием, нежели констатацией квалификации.

Эта статья не является апологией UML. Очевидная перегруженность языка, рыхлое описание и туманная семантика бросаются в глаза. Но нельзя отрицать тот факт, что в UML в унифицированном виде удалось аккумулировать множество разнообразных плодотворных идей, а сообщество носителей языка давно переросло критическую отметку известности. UML является признанным лидером в обсуждаемой области и останется таковым в обозримом будущем. Семантика и нотация UML считаются здесь данностью, известной читателю, и не обсуждаются. Нас интересует, прежде всего, вопрос о том, что и как нужно сделать, чтобы потенциальные достоинства UML дали ощутимые практические результаты. Наиболее важной является инструментальная поддержка UML – чтобы с пользой применять UML для визуального конструирования, нужно иметь адекватные инструментальные средства, поддерживающие UML. В настоящее время инструментальная поддержка UML не вполне удовлетворительна. На рынке

присутствует множество инструментов, и все время появляются новые, но при более пристальном рассмотрении оказывается, что нам предлагают еще одно CASE средство традиционной архитектуры, поддерживающее еще одну неудобную нотацию. По мнению автора, вопрос о разумной поддержке UML не так прост и требует исследования. Именно из-за плохой инструментальной поддержки UML используется в основном для рисования (необязательных!) диаграмм, и не более того. Только при наличии адекватной инструментальной поддержки можно будет надеяться на распространение модельно-орентированного процесса разработки и получение выгод, которые обещает эта идея.

Целью данной работы является исследование вопроса о принципиальной реализуемости и необходимых свойствах инструментальных средств, поддерживающих модельно-центрированную разработку приложений на основе UML.

Визуальное моделирование и конструирование приложений

Прежде всего, охарактеризуем область применимости дальнейших построений. Многообразие различных типов приложений необозримо, и нельзя объять необъятное. Предметная область, характер использования, жизненный цикл приложения и другие факторы оказывают значительное влияние на характер процесса разработки. UML – *унифицированный* язык моделирования, но отнюдь не универсальный и не единый. Утверждать, что UML является панацеей от всех детских болезней программирования, было бы явным преувеличением – «серебряной пули нет» [4].

Для определения области применимости нужно выявить влияние UML на процесс разработки. Существует множество взглядов на процесс разработки – разнообразие подходов и моделей указывает на то, что инженерия программирования (software engineering) является инженерной дисциплиной только по названию, а на практике используются в основном субъективные предпочтения и эмпирические правила. Это обстоятельство позволяет нам использовать здесь для формулировки тезиса о влиянии UML свой взгляд на технологию программирования, опуская для краткости детальные обоснования.

Все процессы, связанные с разработкой, можно разделить на три группы:

1) процессы на уровне организации, т. е. порядок проведения типового проекта (отчетность, бюджет и т. д.);

2) процессы на уровне проекта, т. е. отношения между людьми в процессе разработки (распределение ролей, оперативное планирование и т. д.);

3) процессы на уровне работника, т. е. технология программирования в собственном смысле данного термина.

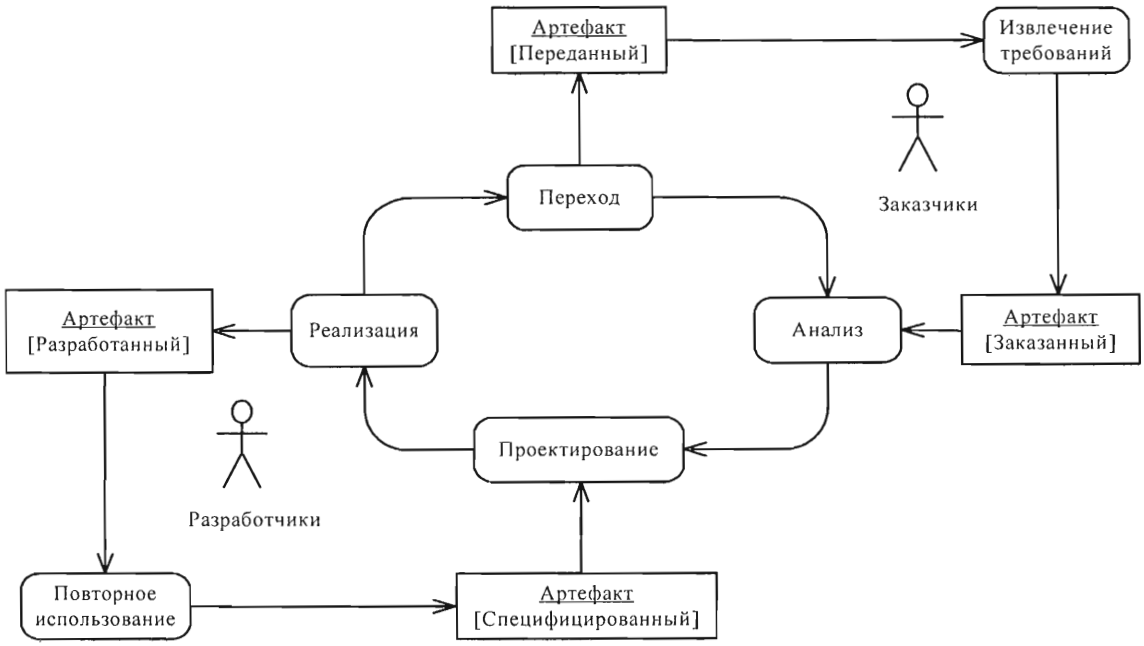
Оценивать качество процессов можно с помощью различных показателей: надежности разрабатываемого программного обеспечения, затрат на разработку, управляемости процесса и т. д. Для различных типов приложений применяются разные системы показателей. Здесь мы рассматриваем заказную разработку отчуждаемых программных продуктов, т. е. предполагается, что в процессе разработки взаимодействуют, по меньшей мере, две стороны, которые мы обозначим устаревшими, но привычными терминами «заказчики» и «разработчики». Кроме того, ограничимся рассмотрением таких проектов, в которых вершины «треугольника качества» (ресурсы, качество, функциональность) не являются жестко фиксированными. Другими словами, мы рассматриваем «типичные» приложения, в которых требования являются достаточно гибкими в определенных пределах: например, допустимо пожертвовать некоторыми не приоритетными функциями, чтобы уложиться в заданные ресурсы по времени, или добавить трудовые ресурсы, чтобы обеспечить приемлемый для заказчика уровень качества. Для таких проектов нас интересует один интегральный показатель – продуктивность, которую здесь мы упрощенно определим как отношение прибыли к трудозатратам. В этих условиях, по субъективным наблюдениям автора, соотношение продуктивности от вложений в улучшение процессов трех выделенных выше групп определяется примерно как 1 : 3 : 10. Другими словами, рубль вложений в технологию программирования на уровне работника дает в десять раз больший прирост продуктивности по сравнению с вложением в «стандартный процесс». Еще раз оговоримся, что все эти утверждения не имеют универсального характера и не являются законами природы – это не более чем эмпирическое обобщение наблюдений автора, и они вряд ли могут быть распространены за пределы оговоренных ограничений.

Что же влияет на продуктивность программирования на уровне работника? По убеждению автора, таких факторов два: 1) сокращение количества внеплановых изменений артефактов; 2) увеличение объема повторно использованных артефактов.

Первый фактор означает уменьшение количества грубых ошибок, допускаемых на фазах анализа и проектирования (прочими ошибками можно пренебречь – стоимость их устранения при использовании современных технологий незначительна). Второй фактор означает увеличение объема повторного использования сделанного ранее. Речь идет о повторном использовании всех видов артефактов: законченных компонентов, фрагментов кода, архитектурных решений и т. д. В частности, применение образцов проектирования мы относим к повторному использованию.

Рассмотрим упрощенную (но не противоречащую наиболее распространенным моделям, например, [5]) схему процесса разработки (рис. 1).

Обсуждение последовательности и названий основных фаз процесса, а также связи между фазами по данным в процессе и условия переходов между



■ *Рис. 1. Циклы повышения продуктивности*

фазами мы оставляем в стороне, поскольку для наших целей это несущественно (в этой статье мы не намерены предлагать вниманию читателей новый «стандартный» процесс).

Выделим два «боковых» цикла движения артефактов в процессе, которые мы называем циклами повышения продуктивности.

1. На стороне заказчика происходит анализ и извлечение требований. В нормальной ситуации требования возникают не «из головы» заказчика, а в результате анализа предметной области и, в частности, анализа использования существующего программного обеспечения. Движение артефактов в этом цикле определяет величину первого фактора повышения продуктивности: неадекватные требования порождают ошибки проектирования, адекватные требования позволяют их избежать.

2. На стороне разработчика вновь созданные (или модифицированные) артефакты подготавливаются для повторного использования: документируются, обобщаются, помещаются в репозиторий и т. д. Движение артефактов в этом цикле определяет величину второго фактора повышения продуктивности: практика показывает, что эффективно повторно использовать удается только те решения, которые были для этого специально подготовлены.

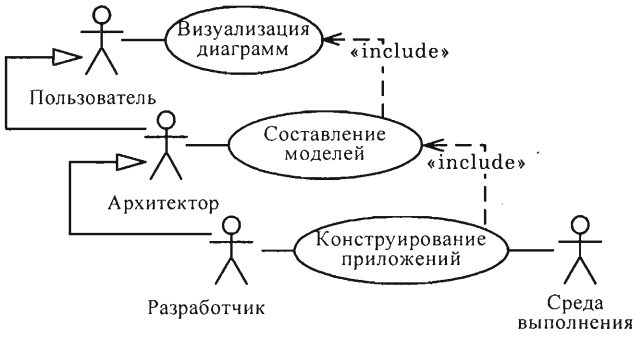
Третий (не указанный на рис. 1) цикл движения артефактов – это стандартный цикл в процессе разработки: требования – проект – реализация – поставляемый продукт.

Наш основной тезис о влиянии UML на процесс разработки состоит в том, что UML позволяет унифицировать представление всех артефактов во всех циклах, и прежде всего в циклах повышения продуктивности. (Заметим, что унификация представления артефактов в основном цикле – это, по нашему мнению, центральная идея модельно-ориентированной разработки.) Если участникам процесса разработки приходится знать и применять десяток (а то и два) различных способов описания артефактов и технологий их создания, то трудно ожидать высокой продуктивности. Действительно, преобразования форматов, изучение специальных приемов, «ручная заточка» инструментов в каждом проекте трудоемки, замедляют скорость движения артефактов и чреваты ошибками. Если же в руках участников процесса есть универсальный инструмент, знакомый всем участникам и адекватный для всех фаз процесса, то индивидуальная производительность (и продуктивность), очевидно, должны иметь тенденцию к росту. В сущности, наш тезис является парафразой известного положения «о пользе чертежей», которое нам представляется бесспорным. Таким образом, UML оказывает положительное влияние на продуктивность процесса разработки, если он действительно применяется, причем в соответствии со своим назначением: «для спецификации, документирования, конструирования и визуализации всех артефактов в процессе разработки программного обеспечения» [5].

Рассмотрим, как соотносится сегодняшняя практика использования UML с декларированным выше назначением языка.

По мнению автора, можно выделить три основных варианта использования UML [9], представленных на рис. 2.

Вариант использования «Визуализация диаграмм» подразумевает изображение диаграмм UML с целью обдумывания, обмена идеями между людьми, документирования и т. п. Значимым для пользо-



■ *Рис. 2. Варианты использования UML*

■ ***Достоинства и недостатки визуального конструирования приложений***

Достоинство	Недостаток
Диаграммы наглядны	Диаграммы не компактны
Высокий уровень абстракции	Плохо выразимы некоторые простые программистские конструкции
Высокий уровень формализации	Сложная для освоения нотация

вателя результатом в этом случае является само изображение диаграмм. Вообще говоря, в этом варианте использования языка поддерживающий инструмент не очень нужен. Иногда рисование диаграмм от руки фломастером с последующим фотографированием цифровым аппаратом может оказаться практичнее.

Вариант использования «Составление моделей» подразумевает создание и изменение модели системы в терминах тех элементов моделирования, которые предусматриваются метамоделью UML[7]. Значимым результатом в этом случае является машиночитаемый артефакт с описанием модели. Для краткости будем называть такой артефакт просто моделью, деятельность по составлению модели – моделированием, а субъекта моделирования – архитектором (потому что соответствующие существительные – модельщик, модельер – в русском языке уже заняты).

Вариант использования «Конструирование приложений» подразумевает детальное моделирование, проектирование, реализацию и тестирование приложения в терминах UML. Значимым для пользователя результатом в этом случае является работающее приложение, которое может быть, например, скомпилировано во входной язык конкретной системой программирования или сразу интерпретировано средой выполнения инструмента. Этот вариант использования наиболее сложен в реализации, но именно он лежит в основе концепции модельно-ориентированной разработки. Термин «конструирование приложений» означает по существу то же самое, что и «разработка приложений». Мы используем этот новый термин, чтобы отличать модельно-ориентированную разработку от традиционных подходов к разработке. Кроме того, термин «конструирование приложений» созвучен термину «construction», который используют в этом контексте авторы UML.

Однако современные инструменты поддерживают указанные варианты использования далеко не в равной степени. Все инструменты умеют (плохо или хорошо) визуализировать все типы диаграмм UML, некоторые инструменты позволяют построить модель, допускающую дальнейшее использование, но только немногие инструменты могут генерировать исполнимый код и отнюдь не для всех диаграмм. Имеется множество практических и организационных причин, по которым указанные варианты использования неравноправны и в разной степени поддержаны в современных инструментах.

Практические причины заключаются в том, что визуальное конструирование наряду с очевидными достоинствами имеет и столь же очевидные недостатки, на которые не стоит закрывать глаза. В таблице указаны наиболее существенные, по нашему мнению, причины.

Достоинства UML не во всех случаях преобладают над его недостатками, поэтому существуют (и, скорее всего, будут существовать) категории пользователей языка, не заинтересованные в поддержке варианта использования «Конструирование приложений». Такие пользователи не согласны платить за ненужные им (и дорогостоящие) возможности. Производители инструментов не вправе игнорировать потребности рынка. (По субъективному впечатлению автора, количество пользователей, жизненно заинтересованных в трех перечисленных выше вариантах использования, соотносится примерно как 10 : 3 : 1. Например, диаграммы в этой статье подготовлены с помощью Microsoft Visio, а этот инструмент и не претендует на визуальное конструирование приложений.)

С другой стороны, довольно распространенным является отношение к UML как к модной графической надстройке над обычным языком программирования «высокого» уровня (речь идет о так называемых языках третьего поколения 3GL), которая имеет частично графический, а не только текстовый синтаксис. Несколько упрощая, можно сказать, что разработка сводится к рисованию диаграммы классов, после чего тела методов вписываются вручную. При таком использовании UML фактически оказывается средством автоматического добавления графических комментариев к программе. Действительно, структура наследования (для небольших проектов) наглядно видна на диаграмме классов, но, по большому счету, этим визуальное моделирование и ограничивается. Мы не склонны считать такую практику визуальным конструированием приложе-

ний, во всяком случае, рассчитывать на значительное повышение продуктивности точно не стоит. UML *можно* использовать как еще один традиционный язык программирования, но, как мы рассчитываем показать читателю, делать этого *не нужно*!

Организационные причины заключаются в том, что процесс стандартизации UML неоправданно затянулся (например, стандарт UML 2.0 находится в состоянии «финализации» уже более года). Данную ситуацию нетрудно объяснить. UML был задуман и определен как язык-оболочка, унифицирующий как можно больше известных и зарекомендовавших себя на практике приемов визуального моделирования и конструирования. Такой подход с необходимостью ведет к сложности и громоздкости языка и его описания. Некоторые полагают, что «UML рухнет под собственной тяжестью» [10]. Мы надеемся показать ниже, что хотя риск такого печального исхода действительно велик, он распространяется в основном на производителей инструментов, но не на пользователей языка, а потому, принимая предупредительные меры, есть надежда избежать худшего.

Сложность и громоздкость языка привели к тому, что подавляющее большинство пользователей знакомится с языком не по оригинальным источникам и утвержденным спецификациям стандартов, которые слишком сложны для восприятия, а с помощью популяризаторов и интерпретаторов, среди которых, к сожалению, немало недобросовестных и некомпетентных людей. Наблюдения автора показывают, что многие пользователи UML владеют языком поверхностно. В результате поставщики инструментов также не утруждают себя следованием стандарту – например, не проверяют правила непротиворечивости, определенные в стандарте [11], не поддерживают (или, хуже того, искажают) стандартные конструкции и канонические диаграммы – да и зачем напрягаться, если имеется в виду только рисование необязательных картинок?

Если сопоставить нынешние инструменты, поддерживающие UML, с традиционными системами программирования, то вариант использования «Рисование диаграмм» соответствует наличию простого текстового редактора, «Моделирование систем» – синтаксическому анализатору, «Конструирование приложений» – генерации кода и наличию административной системы времени выполнения. Языки «программирования», которые не подразумевают выполнения, иногда используются, например, для публикации алгоритмов. Однако чаще производители систем программирования, заявляя о поддержке языка, подразумевают полную поддержку, включая точное соответствие стандартам и многое сверх того, например наличие развитых библиотек готовых к использованию компонентов. В случае с UML ситуация прямо противоположная: заявляя о поддержке UML, некоторые производители инструментов не считают себя связанными серьезными обязательствами. Действительно, правомерно ли утверждать, что инструмент поддерживает описание поведения с помощью UML, если он позволяет *нарисовать* диаграмму состояний и не более того? Между тем даже известнейшие инструменты (например, Rational Rose и Together Control Center), декларируя поддержку UML различных версий, *ничего* не умеют делать с упомянутым базовым средством описания поведения в UML (т. е. с конечными автоматами).

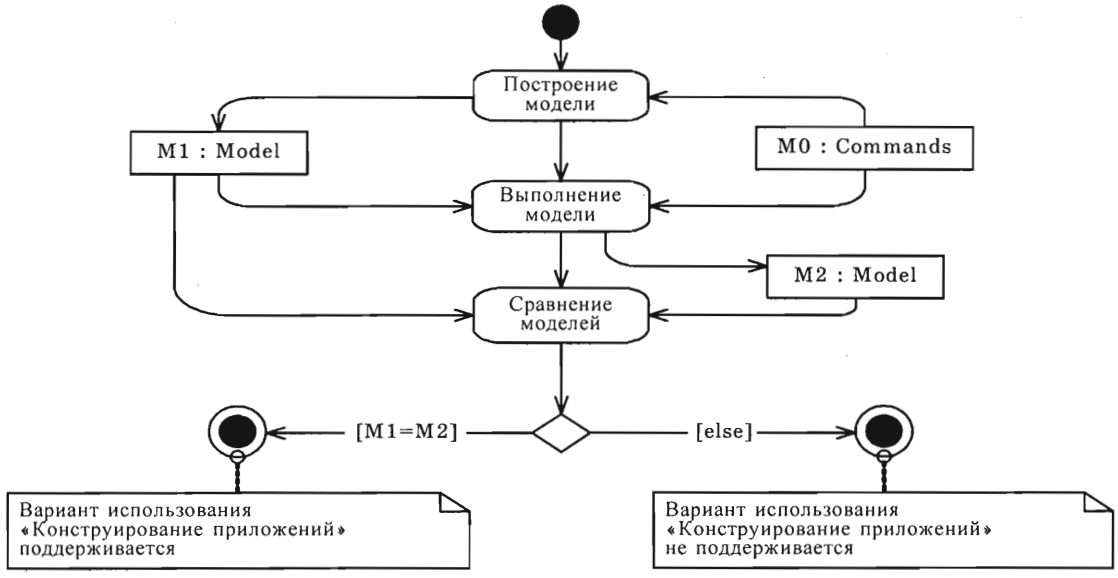
Таким образом, сравнение инструментов по критерию степени поддержки языка расплывчато и субъективно. Предлагаем следующие объективные критерии того, можно ли считать инструмент полностью поддерживающим какой-либо вариант использования UML.

1. Инструмент полностью поддерживает вариант использования «Визуализация диаграмм», если он позволяет отобразить все графические конструкции, предусмотренные описанием нотации в стандарте.

2. Инструмент полностью поддерживает вариант использования «Составление моделей», если он позволяет построить объекты и их комбинации для всех классов метамодели, предусмотренных описанием семантики в стандарте.

3. Инструмент полностью поддерживает вариант использования «Конструирование приложений», если он позволяет преобразовать модель приложения в работающее приложение.

Заметим, что варианты использования очевидным образом включают друг друга, и если инструмент полностью поддерживает разработку, то он, скорее всего, поддерживает также моделирование и рисование. Поставим вопрос: как можно определить, что предложенные критерии выполнены и инструмент действительно поддерживает тот или иной вариант использования? Нетрудно проверить, что инструмент умеет рисовать все, что нужно, – для этого хватит получаса при условии знания языка. Можно проверить, что инструмент создает все элементы модели, предусмотренные стандартом, хотя это и хлопотно – счет тестовых сценариев пойдет на сотни, так как в языке много отдельных конструкций. Но с проверкой третьего варианта использования дела обстоят сложнее. Операционная семантика UML туманна, если не сказать хуже (за что недоброжелатели подвергают язык постоянной и справедливой критике). В то же время, по нашему мнению, плохое определение или даже отсутствие операционной семантики в стандарте UML еще не является непреодолимым препятствием для реализации варианта использования «Конструирование приложений». Язык устроен таким образом, что семантику можно *доопределить* прямо в инструменте. Стандартные механизмы расширения и точки вариации семантики оставляют достаточный простор. Похожая проблема (но менее остро) стоит и перед разработчиками традиционных систем программирования. Обычно ее решают чисто практически: договариваются о представительном наборе примеров, и если прогон примеров дает ожидаемые результаты, то система программирования считается приемле-



■ *Рис. 3. Сценарий проверки варианта использования «Конструирование приложений»*

мой. Однако стандартного набора примеров визуальной разработки пока нет, и приходится их выбирать и придумывать.

Традиционно в качестве теста для инструмента разработки используют сам этот инструмент разработки. Например, язык программирования хорош, если на этом языке можно написать хороший компилятор данного языка. Такой прием обычно называют раскруткой (bootstrapping). Мы считаем достаточно представительным примером приложения сам инструмент визуального конструирования. Таким образом, окончательно наш критерий можно сформулировать следующим образом: *инструмент полностью поддерживает* UML, *если он допускает полную раскрутку.* Другими словами, если на вход инструмента подать модель инструмента, созданную в этом инструменте, то на выходе мы получим тот же инструмент (тот же в функциональном смысле, т. е. так же работающий). Проверить это (до некоторой степени) можно с помощью сценария, приведенного на рис. 3. Здесь предполагается, что инструмент умеет выполнять последовательности команд и что процесс построения любой модели можно описать такой последовательностью команд (трудно представить себе обратное). Тогда, если инструменту подать последовательность команд, описывающую построение его модели (на рис. 3 она обозначена M0), то инструмент должен построить некоторую модель самого себя (M1). Если теперь выполнить модель M1 (запустить интерпретацию или же сгенерировать код и запустить его) и еще раз подать на вход этому сеансу выполнения ту же последовательность команд M0, то должна быть построена еще одна модель (M2). Если окажется, что M1 и M2 совпадают, то, по нашему мнению, это позволяет с достаточной степенью уверенности считать, что инструмент функционально тождественен своей модели и вариант использования «Конструирование приложений» поддерживается.

Отметим, что в настоящее время не существует инструментов, полностью поддерживающих UML согласно приведенному критерию.

Современные тенденции в методологии программирования

Рассмотрим некоторые концепции методологии программирования, о которых точно известно, что их применение в практическом программировании положительно влияет на продуктивность. Обозначим их как компонентно-ориентированное программирование, предметно-ориентированное программирование и аспектно-ориентированное программирование.

Мы не собираемся детально излагать эти концепции – предполагается, что они хорошо известны читателю (возможно, под другими названиями, поэтому основные идеи во избежание недоразумений мы все-таки повторим). Наша цель состоит в том, чтобы установить, насколько адекватен UML, лежит ли этот язык в русле основных тенденций или «идет против течения».

Компонентно-ориентированное программирование старо, как само программирование, если не старо как мир, поскольку является реализацией принципа «разделяй и властвуй». Во всяком случае, указать первоисточник этой идеи затруднительно. Сама же идея очевидна: целесообразно конструировать программы не из мелких деталей (операторов языка программирования), а из более крупных блоков (компонентов). Выгоды очевидны: составить большую программу из готовых компонентов намного легче, чем написать «с нуля» (такую же по возмож-

ностям) программу; готовые компоненты являются тем, что можно и нужно повторно использовать.

Понятие «компонент» довольно расплывчатое, были предложены и использованы многочисленные технологии и механизмы, поддерживающие компонентное программирование, и компоненты в них понимаются и описываются немного по-разному. Упрощенно все разнообразие видов программных компонентов можно поделить на три типа:

1) неструктурированные фрагменты исходного кода на языке программирования;

2) структурные части исходного кода программ (их часто называют модулями);

3) скомпилированные исполнимые части программ (например, библиотеки динамического связывания).

В первом случае решение прямолинейное и понятное, но многого ожидать от него не приходится: если текст программы собирается из компонентов путем копирования и вставки, то очень вероятны ошибки, связанные с «настройкой по месту» скопированного фрагмента. Впрочем, в отношении компонентного программирования методом копирования и вставки инструменты, основанные на UML, не хуже других: фрагменты моделей можно также легко копировать и вставлять, как и тексты.

Компоненты на уровне модулей исходного кода удобны, если используется один язык программирования с хорошо определенным понятием модуля (например, Pascal или Java). Если же требуется использовать компоненты, написанные на разных языках и для разных платформ, то возникают проблемы. В UML имеется хорошо определенное понятие модуля (пакет), а поскольку язык по самой своей сути претендует на независимость от платформы, число и сложность проблем при сборке моделей из пакетов UML оказываются меньше. Заметим, что при описании UML 2.0 [8] введена и активно используется для описания самого языка своеобразная операция слияния пакетов («merge»). Этот вид компонентного программирования специфичен для модельно-центрированной разработки. Трудно представить себе столь же избирательную операцию по сборке на уровне текстов. Опуская детали, можно сказать, что слияние позволяет использовать пакет как компонент, извлекая из него только то, что нужно при сборке модели. Таким образом, компонентное программирование на уровне исходных кодов поддержано в UML по меньшей мере не хуже, чем у конкурентов.

Компонентное программирование на уровне исполнимых компонентов имеет наибольшее практическое применение в настоящее время. Разрабатываются и применяются весьма изощренные механизмы, призванные обеспечить компонентное программирование на объектном уровне. Наиболее яркий пример последнего времени – это, несомненно, .NET [12]. Общая среда выполнения, обмен метаинформацией и другие механизмы обеспечивают достаточно гибкие возможности сборки программ из компонентов. Заметим, что разработчики .NET были связаны искусственным ограничением, состоящим в том, что исходный код компонента считается недоступным во время использования компонента.

По мнению автора, защита интеллектуальной собственности разработчиков путем сокрытия исходных кодов программ – это укоренившийся, но общественно вредный анахронизм. Практика сокрытия исходного кода невыгодна самим разработчикам. Действительно, сокрытие исходного кода затрудняет его повторное использование и упущенная выгода намного превышает ущерб от злонамеренного нарушения авторских прав. Кстати, в современной открытой информационной среде контрафактное использование кода программ технически вполне возможно выявить и пресечь. В модельно-центрированной разработке сокрытие моделей – нонсенс, противоречащий самой сути подхода. Общественные движения, направленные на расширение публичного доступа к информации, экономически обоснованы, и потому победа будет за ними. В этой связи, по нашему мнению, особого упоминания заслуживает движение за открытую проектную документацию [13], поскольку оно в наибольшей степени соответствует по духу модельно-ориентированной разработке.

Возвращаясь к компонентному программированию на UML, отметим следующее. Диаграммы компонентов UML предоставляют достаточно наглядные средства для описания состава и структуры взаимодействия компонентов (особенно с учетом симметричной нотации для требуемых и предоставляемых интерфейсов, введенной в UML 2.0), а диаграммы взаимодействия являются достаточным ресурсом для описания самого взаимодействия. Важным, на наш взгляд, является появление в UML 2.0 специального вида машин состояний, описывающих протокол взаимодействия компонентов.

Мы видим, что применение UML ничуть не противоречит компонентному программированию в любом его толковании. Более того, мы рискнем предположить, что применение UML в ближайшем будущем позволит рассматривать еще одно толкование понятия «компонента»: компонент – это экземпляр образца проектирования, т. е. решение задачи путем применения образца проектирования в конкретном контексте.

Подводя итоги обсуждения компонентного программирования, подчеркнем еще раз, что оно явно поощряет повторное использование, а потому ведет к повышению продуктивности. Приятно отметить, что в качестве средства компонентного программирования UML не только не отстает, но даже несколько опережает распространенную сегодня практику.

Обратимся теперь к предметно-ориентированному программированию. Основная идея здесь также банальна: программировать нужно в терминах конкретной предметной области, тогда программы для этой предметной области будут простыми, понятными и надежными. Мы не будем обсуждать здесь

предметно-ориентированное программирование во всех деталях, тем более что они намного отличаются у разных авторов. Очень краткий, но емкий обзор приведен в работе [14]. Отметим только одно обстоятельство, важное в контексте нашего рассмотрения. Программировать в терминах предметной области могут и должны специалисты в этой предметной области, которые, хотя, может быть, и не являются профессиональными программистами, но показывают очень высокую продуктивность. Личные наблюдения автора полностью согласуются с этим тезисом, более того, у нас есть объяснение: если в циклах повышения продуктивности (см. рис. 1) заказчики и разработчики – одно действующее лицо, то движение артефактов заметно ускоряется. С других исходных позиций, но к тому же выводу приходят экстремальные программисты, настаивая на постоянной вовлеченности заказчика в процесс разработки.

Чтобы оценить пригодность UML для предметно-ориентированного программирования, заметим следующее. Обычно апологеты предметно-ориентированного конструирования приложений акцентируют внимание на выразительности предметно-ориентированных программ и простоте их составления, но это только верхушка айсберга! Квалифицированная формализация предметной области, разработка развитых пакетов прикладных программ для нее, изобретение и реализация специфических интерфейсных средств остаются как бы за кадром. Системы предметно-ориентированного программирования просты в использовании, но сложны в реализации. Например, известная программа Excel, которую мы считаем классическим примером предметно-ориентированной системы программирования, намного превосходит по богатству возможностей современные системы программирования на обычных универсальных языках программирования. Сложную задачу по обработке числовых данных, которую опытный пользователь Excel решает «в два счета» с помощью сводных таблиц и многочисленных надстроек Excel (например, надстройка «поиск решения»), команда обычных профессиональных программистов на C++, скорее всего, вообще не сможет запрограммировать за разумное время – не хватит квалификации в предметной области.

Многие справедливо отмечают, что UML слишком сложный язык, и конечному пользователю – специалисту в предметной области – было бы трудно построить содержательную модель так же легко, как это делается с помощью специализированного предметно-ориентированного языка. Критика направлена мимо цели. Во-первых, построить модель предметной области на C++ ничуть не проще. Во-вторых, делать этого не нужно. Действительно, зачем насильно втискивать предметную область в UML? Наоборот, целесообразно воспользоваться гибкостью UML и создать специализированный предметно-ориентированный язык, на котором модель запишется легко и просто. И здесь UML может, по нашему мнению, составить конкуренцию традиционным подходам. Главным конкурентным преимуществом UML при этом является возможность метамоделирования.

Метамоделирование – это возможность изменять метамодель, т. е. тот язык (включая семантику), на котором пользователь строит свои модели. Поскольку мы рассматриваем инструмент, которому полностью подвластна его собственная модель (метамодель), то возможности метамоделирования на UML намного богаче, чем на обычных языках программирования. Поясним примером: обычная система программирования, может быть, позволяет переопределить стандартную библиотеку встроенных функций, меняя до некоторой степени семантику языку, но вряд ли позволит вмешиваться в протокол вызова функций или в механизм проверки типов. Для инструмента, полностью поддерживающего UML, все это возможно.

Последняя тенденция, которую мы хотим затронуть, – аспектно-ориентированное программирование. Это сравнительно новое изобретение, но основанное на том же повторном использовании кода. Идея, как обычно, проста и базируется на следующем наблюдении: большая часть кода в реальных программах имеет рутинный характер – это повторы практически одинаковых фрагментов, написание которых требует не глубоких раздумий, а только сугубого внимания. Ярким примером, на котором часто объясняют аспектно-ориентированное программирование, является ведение программой протокола своего выполнения. Проблема состоит в том, что похожие операторы записи в файл протокола (или вызов соответствующей процедуры) должны быть помещены в сотни разных мест исходной программы. Эти операторы, может быть, зависят от контекста, в котором выполняется протоколирование, но, в общем, очень похожи. Идея состоит в том, чтобы написать заготовку нужных операторов один раз («определить аспект»), а также один раз определить правило модификации аспекта в соответствии с контекстом и один раз определить правило нахождения мест в тексте программы, куда эти операторы должны быть вставлены. Система аспектно-ориентированного программирования автоматически вставит нужные операторы везде, где требуется. Экономия трудозатрат, повышение надежности, улучшение читабельности, удобство повторного использования очевидны.

Классические системы программирования реализуют идею аспектов с помощью специального модуля (препроцессора или прохода транслятора), который обычно называют «ткач», поскольку этот модуль как бы «вплетает» аспекты в основной код программы. Насколько же UML приспособлен для реализации этой идеи? Заметим, во-первых, что в UML имеется зависимость со стандартным стереотипом «extend», которая прямо реализует идею расширения варианта использования «извне». Правда, такая зависимость предусмотрена только для вариан-

тов использования, но можно определить соответствующие стереотипные зависимости и для других элементов моделирования. Мы хотели бы обратить внимание на то, что можно пойти гораздо дальше. В сущности, аспекты – это пример *нелокальной* операции над текстом программы (другой пример – так называемый «рефакторинг»). Такого рода нелокальные операции с текстами не слишком широко распространены и применяются реже, чем они того заслуживают, из-за сугубой бедности структуры текста программы. В лучшем случае мы можем опираться на древовидную синтаксическую структуру. Между тем в модели UML, особенно при наличии средств метамоделирования, для формулировки правил нелокальных преобразований модели доступно значительно больше информации: наряду с отношением владения между элементами модели, которое соответствует обычной синтаксической иерархии, можно использовать причинно-следственные связи, выраженные зависимостями со стереотипом «refine», строгую типизацию элементов, специфические отношения на диаграммах и многое другое. Пример подобного преобразования модели – поиск и выделение в конечном автомате (диаграмме состояний) составных состояний, которые могли бы объединять состояния исходного автомата. Обсуждение подобных нелокальных систематических преобразований модели только начинается: здесь пока больше неясных вопросов, чем готовых ответов.

Таким образом, мы видим, что концепции и средства UML в основном хорошо согласованы с современными тенденциями в программировании, и дело за тем, чтобы обеспечить должную инструментальную поддержку.

Общие требования к инструменту визуального конструирования

Первое требование к инструменту визуального конструирования состоит в том, что инструмент должен отвечать своему названию, т. е. он должен позволять конструировать приложения. В первом разделе статьи мы сформулировали критерий того, что инструмент поддерживает вариант использования «Разработка приложений» – инструмент должен быть достаточен для воспроизведения себя самого по своей модели. Это требование главное, но не единственное.

Другие требования вытекают из объективных обстоятельств и ограничений. Заметим, что мы обсуждаем инструменты, поддерживающие язык UML, который, в свою очередь, является предметом международной стандартизации. Далее, напомним, что визуальное конструирование не только академически интересно, но и нацелено на повышение продуктивности разработки, а значит, инструмент должен быть согласован с современными тенденциями методологии программирования. Для определения вытекающих требований целесообразно использовать тот же принцип самоприменимости. Отсюда следует, что инструмент на своем примере должен демонстрировать следование принципам компонентного, предметно-ориентированного и модельно-орентированного программирования. (Аспектно-ориентированное программирование мы не включили в число основных требований, так как оно полезно, но не первостепенно.) Таким образом, наши требования к инструменту: соответствие стандартам, компонентная архитектура, предметная ориентация, модельная ориентированность.

Обсудим эти требования чуть подробнее. Стандартизация, безусловно, полезна, а в инженерных областях – необходима. Но само по себе формальное соответствие стандартам еще не является существенной ценностью для пользователей инструмента. Стандарт языка нужен для обеспечения унифицированного использования, совместимости разных инструментов и переносимости на различные платформы. Выше мы отметили, что возможность метамоделирования ведет к высокой гибкости инструмента визуального конструирования, намного превосходящей модифицируемость традиционных систем программирования. Особенности (например, механизмы расширения и точки вариации семантики) и, к сожалению, недоговоренности стандартного определения языка ведут к появлению диалектов. Модели, созданные разными разработчиками и с помощью различных инструментов, могут быть совершенно не похожи друг на друга, и мы не видим в таком разнообразии ничего плохого, если модели отвечают основному назначению: визуализации, спецификации, конструированию и документированию программных систем. Применительно к UML разумно выдвинуть требование о соответствии скорее духу, нежели букве стандарта: язык инструмента не должен *противоречить* стандарту (но может быть расширен и доопределен в нужных инструменту направлениях). Что касается совместимости различных инструментов, то владелец процесса стандартизации UML – Object Management Group – конечно, уделяет внимание этому вопросу, но явно недостаточное. В качестве средства обеспечения совместимости предлагается использовать XMI (XML Metadata Interchange) – специальное приложение XML, предназначенное для этой цели. Но со стандартизацией XMI дело обстоит еще хуже, чем с UML – отставание версий XMI от UML, несоответствие версий и т. д. Реальные промышленные инструменты кое-как понимают друг друга, но с явными ошибками. Впрочем, производителей инструментов это не очень беспокоит: каждый из них претендует на то, чтобы стать единственным нужным пользователю поставщиком инструментов моделирования. Требовать невозможного неразумно: достаточно обеспечивать совместимость в тех пределах, в которых (будет) определен XMI. Наконец, многоплатформенность для самого инструмента необязательна, так как одна из основных идей MDA – это кроссплатформенная генерация кода. Не стоит требовать, чтобы инструмент работал в любой среде –

достаточно, чтобы умел конструировать приложения для выполнения в *нужной* среде.

Обратимся теперь к требованию компонентной архитектуры. Прежде всего, констатируем, что инструмент компонентного конструирования должен быть примером компонентного конструирования, а значит, должен быть не «монолитным» программным продуктом, а конструктором инструментов. Другими словами, мы выдвигаем требование не просто компонентой архитектуры, а динамически изменяемой (расширяемой) пользователем. В любой момент компоненты могут быть изменены, удалены или добавлены в соответствии с потребностями конкретного случая применения инструмента. Такое требование не является чем-то исключительным: многие современные приложения используют идею надстроек. Надстройка – это компонент, реализующий специфические функции и включаемый в состав приложения по желанию пользователя. Особенностью выдвигаемого здесь требования является то, что инструмент должен быть инструментом разработки своих надстроек.

Обсудим теперь требование предметной ориентации. Предметная ориентация подразумевает использование специального языка (или языков), ориентированного на предметную область. В данном случае предметной областью является метамоделирование, и соответствующий предметно-ориентированный язык – MOF (Meta Object Facility) [16] – уже предложен той же организацией, что и UML. Однако применительно к MOF можно повторить сказанное об XMI – ход процесса стандартизации отстает от потребностей практики. В результате во многих ведущих проектах используются свои, альтернативные варианты предметно-ориентированных средств метамоделирования [17]. Поэтому мы не склонны формулировать наше требование предметной ориентации в жестком формальном виде «MOF Compliant». Любые средства метамоделирования допустимы, если они достаточны и удобны. Можно использовать и свою метамодель, и свой язык, даже текстовый!

Наконец, последнее наше общее требование – модельная центрированность – является, по-видимому, самым важным, необычным и трудновыполнимым. Необычность проявляется даже в том, что сама формулировка требования режет слух. Мы используем буквальный перевод английского термина model centric просто потому, что пока не нашли более подходящего русского перевода. Речь идет о том, что модель является важнейшим и (в идеале) единственным абсолютно необходимым артефактом процесса разработки. Поясним это требование противопоставлением. При разработке приложения с помощью традиционной системы программирования исходный код программы имеет решающее значение. В процессе разработки могут использоваться какие-то другие типы артефактов (текстовые описания, диаграммы, тестовые примеры и т. п.), но исходный код является главным: нет кода – нет и разработанного приложения, и напротив, есть работающий код – есть и приложение. Все остальное необязательно. При модельно-орентированной разработке роль кода играет модель, наличие исполнимой модели является необходимым и достаточным условием, а исходный код на традиционном языке программирования – только возможным, но необязательным артефактом разработки. Отсюда следует, что все в инструменте должно быть представлено в форме модели UML, но отнюдь не следует, что в этой модели не должно быть ничего, кроме «стрелочек и квадратиков». Так же, как и любая другая система программирования, система визуального конструирования опирается на набор предопределенных примитивов. По нашему мнению, эти примитивы отнюдь не обязаны ограничиваться только теми понятиями, через которые (фрагментарно!) определена операционная семантика UML – их явно недостаточно. В качестве примитивов можно использовать любые готовые компоненты любой сложности, лишь бы они были описаны в модели, а реализованы они могут быть вовне произвольным образом. В этой связи заметим, что вопрос об алгоритмической полноте UML в контексте нашего рассмотрения представляется хотя и любопытным, но достаточно праздным – напомним, что нашей главной целью является повышение продуктивности разработки программного обеспечения, а не построение еще одной модели вычислимости, тем паче «единой теории программирования».

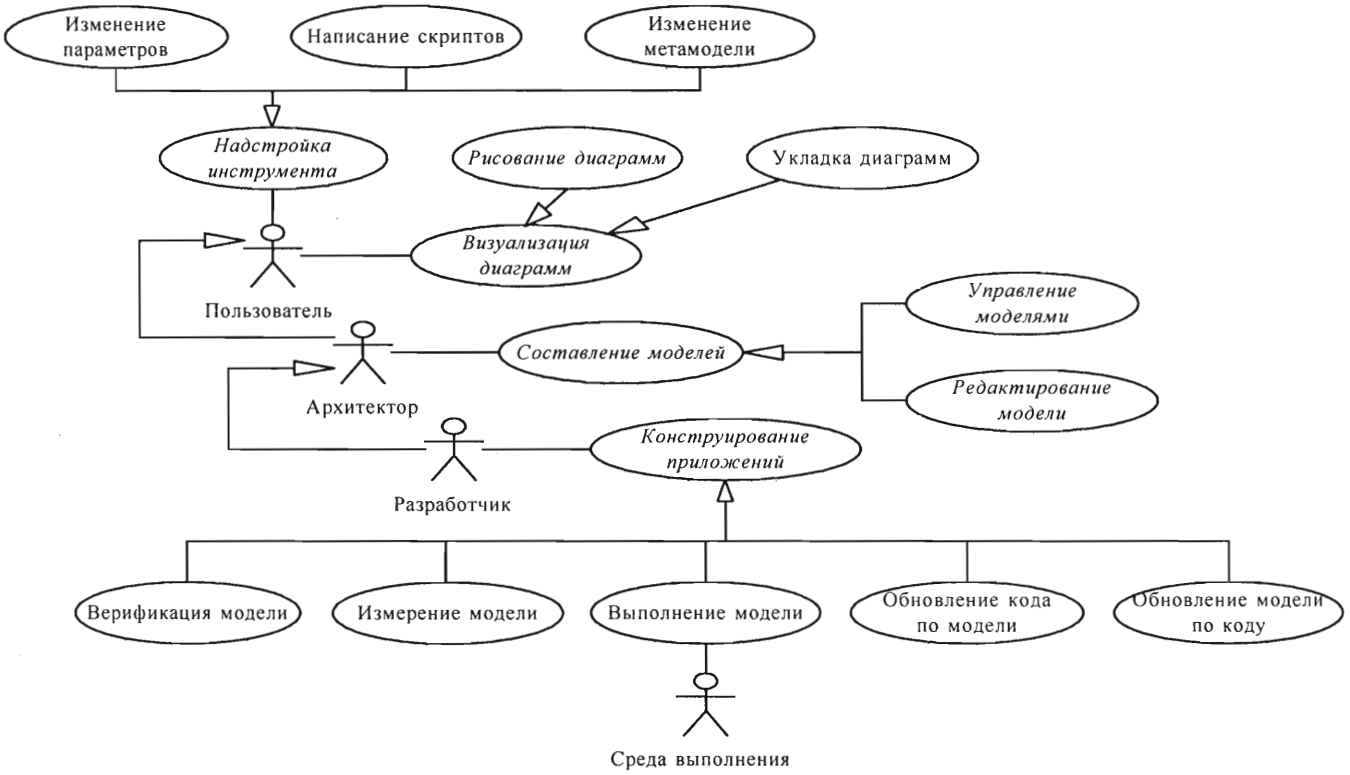
Принципы реализации инструмента визуального конструирования

Имея поставленную цель, сформулированный критерий ее достижения и требования к решению, попробуем определить, что и как нужно сделать для достижения цели.

Прежде всего, следует определить, *что* нужно сделать. Мы, вслед за А. Якобсоном, считаем, что состав компонентов (надстроек) должен управляться вариантами использования. Например, на рис. 4 представлена одна из возможных моделей использования инструмента, полученная уточнением исходной модели (см. рис. 2).

По поводу модели использования необходимо сделать два важных замечания, относящихся к процессу разработки и специфичных для визуального конструирования приложений.

Во-первых, модель использования действительно управляет процессом визуального конструирования, но отнюдь не обязательно является описанием компонентной структуры и состава реализуемых надстроек. Вариант использования может однозначно соответствовать отдельному компоненту или надстройке, но чаще однозначного соответствия не наблюдается: компонент поддерживает несколько вариантов использования, а вариант использования реализуется в нескольких компонентах. В этом случае вариант использования следует считать скорее описанием аспекта, «размазанного» по структуре кода.



■ *Рис. 4. Уточненная модель использования инструмента*

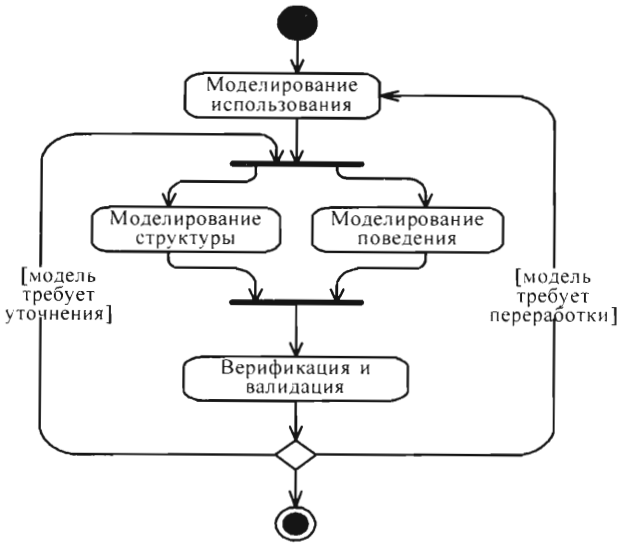
Во-вторых, при визуальном конструировании модель использования не является жестко фиксированной, подобно тому, как не являются жестко фиксированными требования в традиционном процессе разработки. Собственно, модель использования – это и есть требования. Поэтому отдельные процессы и инструменты для управления требованиями не нужны, поскольку требования, по крайней мере, функциональные, являются частью основного артефакта – модели приложения – и управляются совместно с остальными элементами модели. Итеративный процесс визуального конструирования в целом можно представить диаграммой (рис. 5).

Блоки деятельности на рис. 5 названы не совсем привычно, и это отражает убеждения автора: внедрение визуального конструирования окажет серьезное влияние на привычный процесс разработки. Место анализа требований займет моделирование использования; проектирование и реализация сольются в моделирование; отладка и тестирование претерпят радикальные изменения, превратившись в верификацию и валидацию моделей. Но изменятся не только названия фаз процесса – это второстепенно. Важнее то, что кардинально изменится соотношение трудозатрат на разные фазы в процессе. Львиную долю будет занимать проектирование (в форме визуального конструирования), а непродуктивные сейчас процессы отладки, тестирования и документирования перестанут довлеть над менеджерами программных проектов, систематически недооценивающих трудозатраты при составлении планов-графиков.

Как же построить такой инструмент визуального конструирования? Как обычно – постепенно, итерационно (с помощью «инкрементального» процесса разработки). Лучше сразу начать раскручивать инструмент, концептуально ориентированный на визуальное конструирование приложений, а не только на рисование диаграмм. Многие из «больших» инструментов визуального моделирования и проектирования, которые сейчас заявляют о своей поддержке UML 2.0, MDA и визуального конструирования, на самом деле изначально не были ориентированы на эти новые идеи. Конечно, эти инструменты прогрессируют в данном направлении, но лишь с той скоростью, с которой позволяет богатый (и тяжелый!) груз унаследованных компонентов.

Новые проекты выглядят динамичнее. Среди последних наибольшего внимания, по нашему мнению, заслуживает проект UniMod. Автор непосредственно наблюдает развитие проекта, и скорость роста возможностей этого инструмента впечатляет.

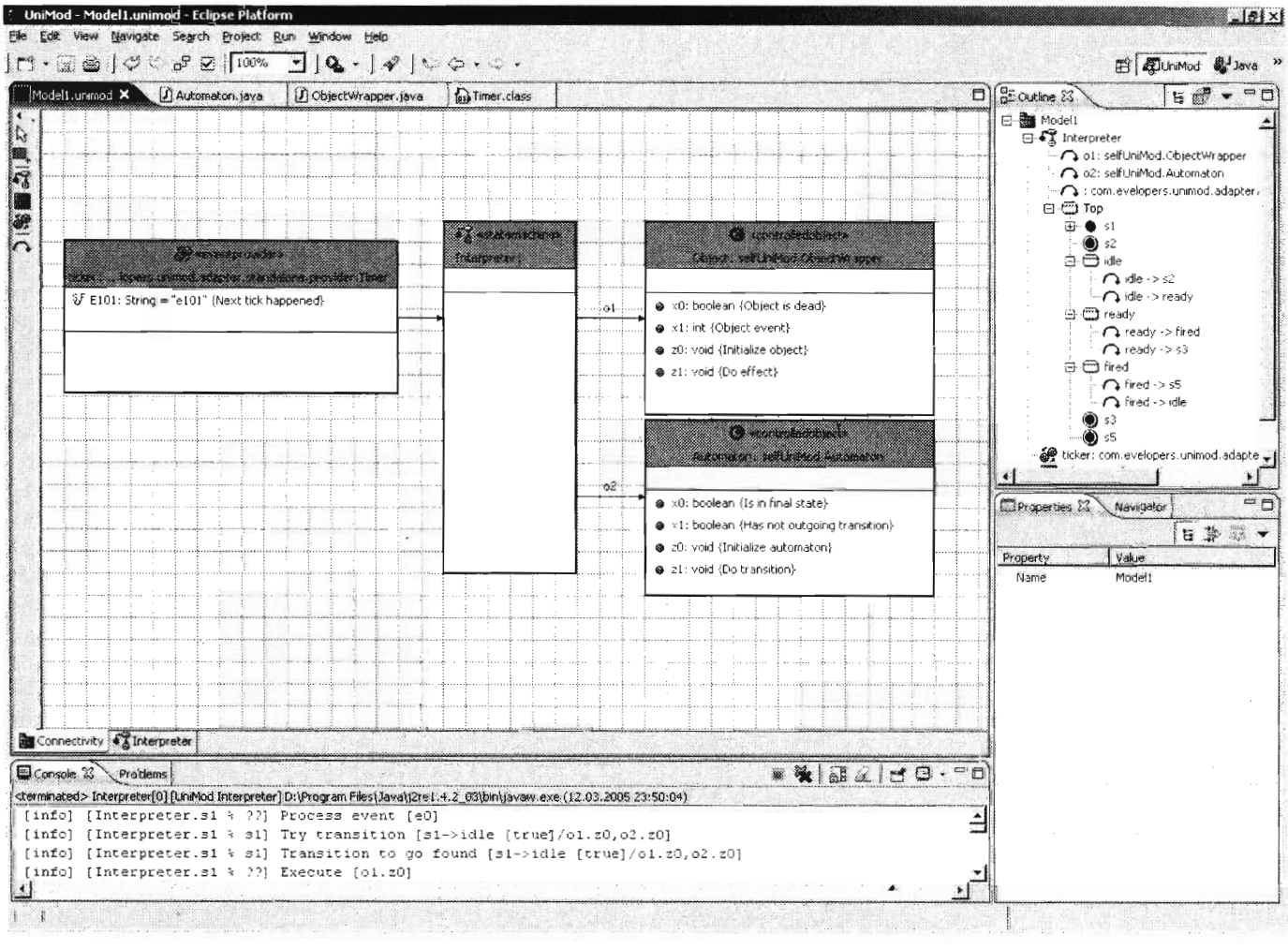
Проект UniMod опирается на ясную парадигму – автоматное программирование [20]. В соответствии с парадигмой автоматного программирования приложение мыслится как совокупность некоторых объектов предметной области и управляющих ими конечных автоматов. Объекты управления могут выполнять некоторые действия, и в них могут происходить события, автоматы при поступлении со-



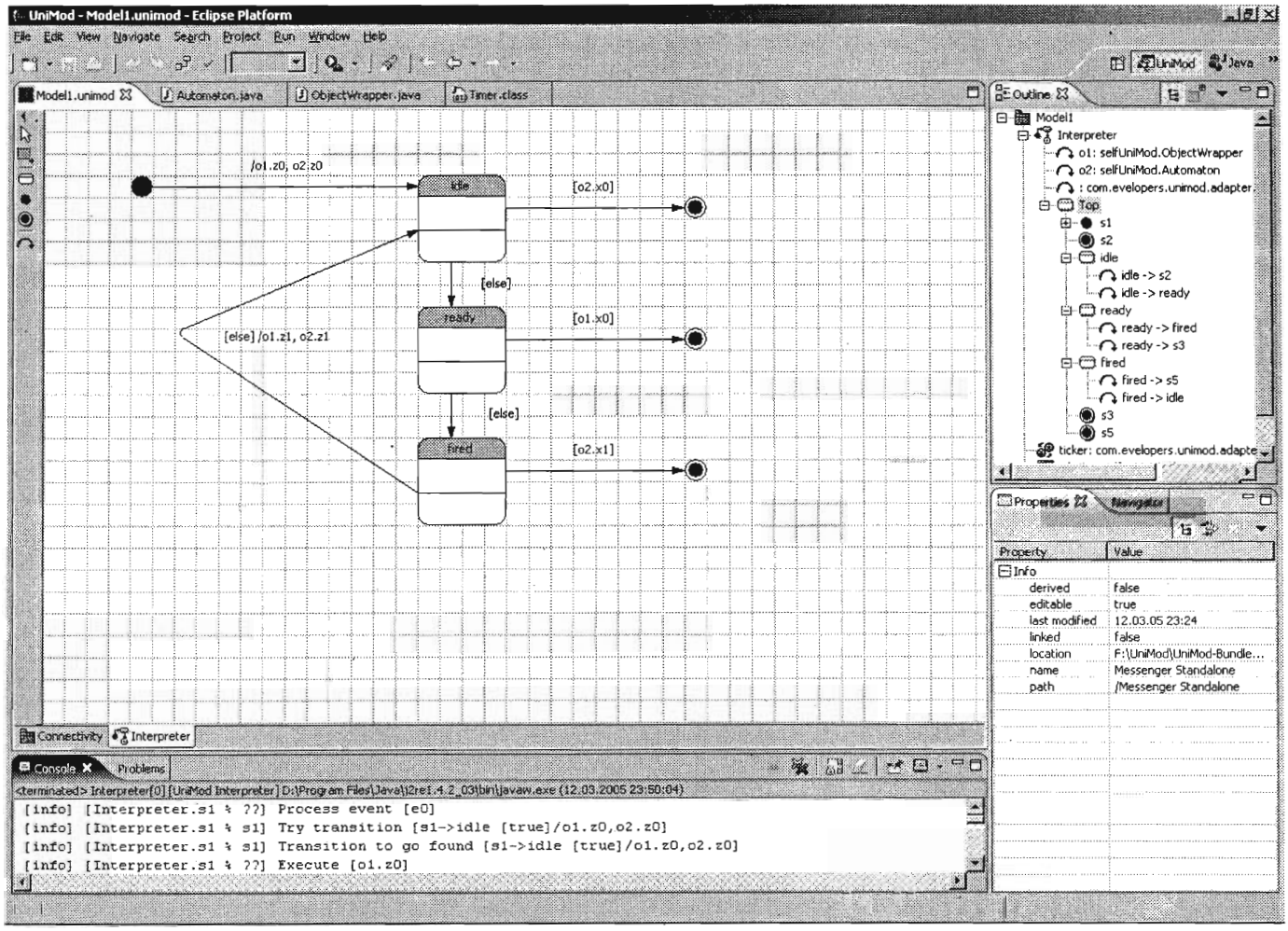
■ *Рис. 5. Итеративный процесс визуального конструирования*

бытий вызывают соответствующие действия объектов. Эта простая и ясная парадигма является необычайно широко применимой. В данной статье автор не намерен отстаивать и разъяснять концепцию автоматного программирования (поскольку эта идея представляется бесспорной), отсылая читателя за блестящими аргументами и многочисленными примерами к первоисточникам [21]. Для нас достаточно того, что автоматное программирование удовлетворяет главному критерию полной раскрутки. В качестве показательного примера мы предлагаем вниманию читателя автоматную модель интерпретатора автоматных моделей.

На рис. 6 и 7 показана упрощенная модель интерпретатора моделей UniMod, реализованная на UniMod. В качестве исходного материала использовано описание работы интерпретатора UniMod, взятое из работы [22]. Для краткости здесь приняты существенные, но не принципиальные ограничения: не рассматриваются составные состояния и вложенные автоматы (а значит, опущена



■ *Рис. 6. Диаграмма связей интерпретатора*



■ *Рис. 7. Диаграмма состояний интерпретатора*

проблема хранения контекста выполнения автомата), интерфейс объектов управления унифицирован (чего нетрудно достичь на практике с помощью соответствующей оболочки) и объекты управления немедленно реагируют событиями на управляющие воздействия.

Концептуально модель приложения в UniMod состоит из диаграмм двух типов:

1) специализированных диаграмм классов, показывающих участвующие в приложении источники событий, объекты управления и автоматы (такие диаграммы авторы UniMod называют диаграммами связей);

2) диаграмм состояний, описывающих поведение автоматов.

Диаграмма связей для упрощенного интерпретатора моделей представлена на рис. 6. Диаграмма выполнена с использованием стилистических соглашений UniMod: слева источники событий, справа объекты управления, а в центре автоматы. Имена выбираются в соответствии с очень жесткой дисциплиной имен, принятой в UniMod. Объекты полагается именовать «o1», «o2» и т. д.; управляющие воздействия – «z0», «z1»; имена событий начинаются с буквы «e»; имена функций, участвующих в сторожевых условиях на переходах, – с буквы «x».

Такая дисциплина имен кажется непривычной и даже архаичной, но у нее есть важные для визуального конструирования достоинства. Во-первых, имена получаются короткие, а значит, даже сложные выражения легко размещаются, например, на стрелках переходов диаграмм состояний. Во-вторых, имена типизированы на лексическом уровне, что очень важно для читабельности диаграмм. Такой стиль действительно противоречит якобы общепринятому правилу обязательного использования длинных содержательных идентификаторов. Заметим, однако, что содержательность легко достигается комментариями и всплывающими подсказками, и при этом не требуется использовать уродливых паллиативов наподобие «венгерской нотации».

Структура связей интерпретатора предельно проста: интерпретатору нужен тактовый генератор, который бы побуждал его работать (на рис. 6 слева), и два объекта управления. Первый из них пред-

ставляет объект или объекты предметной области (на рис. 6 обозначен o1), а второй – сам интерпретируемый автомат (o2).

Диаграмма состояний интерпретатора (рис. 7) сделана как можно более простой, с тем, чтобы еще раз показать на примере, что интерпретация моделей не является чем-то запредельно сложным.

Фактически на этой диаграмме нет устойчивых состояний, поскольку все переходы спонтанны. Таким образом, данная диаграмма может быть прочитана как диаграмма деятельности, и в таком прочтении хорошо видна ее связь с первоисточником, заимствованным из работы [22].

Из приведенного примера видно, что для визуального конструирования совсем не обязательно «поддерживать» весь язык UML. Например, диаграммы связей UniMod являются весьма узким и к тому же специализированным подмножеством диаграмм классов UML. С другой стороны, если некоторая концепция поддерживается, то она должна поддерживаться в полной мере, например так, как поддержаны диаграммы состояний в UniMod.

Заключение

Визуальное конструирование программ возможно, более того, оно реально осуществимо здесь и сейчас.

Помимо теоретических рассуждений и благих пожеланий имеются конкретные примеры инструментов, поддерживающих визуальное конструирование приложений. Самым показательным из известных примеров автор считает UniMod. Будучи с самого начала ориентированным на визуальное конструирование приложений, данный инструмент сконцентрирован на главном – исполнимой визуальной модели приложения.

Статья подготовлена при поддержке «Лаборатории автоматного программирования», организованной корпорацией Borland и Санкт-Петербургским государственным университетом информационных технологий, механики и оптики.

Литература

1. **Брауде Э.** Технология разработки программного обеспечения. СПб.: Питер, 2004.
2. http://www.omg.org/mda/
3. **Буч Г., Рамбо Д., Якобсон А.** UML: специальный справочник. СПб.: Питер, 2002.
4. **Брукс Ф.** Мифический человеко-месяц. М.: Символ, 2000.
5. **Якобсон А., Буч Г., Рамбо Дж.** Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002.
6. **Буч Г., Рамбо Д., Якобсон А.** Язык UML. Руководство пользователя. ДМК, 2000.
7. OMG Unified Modeling Language Specification. Version 1.4. http://www.omg.org
8. http://www.uml.org/#UML2.0
9. **Фаулер М., Скотт К.** UML. Основы/ 2-е изд. М.: Символ, 2002.
10. **Thomas D.** UML – unified or universal modeling language?// Journal of Object Technology. 2003. № 1.
11. **Андреев Н. Д.** Автоматическая верификация модели UML. БИКАМП, 2003.
12. **Уоткинз Д., Хаммонд М., Эйбрамс Б.** Программирование на платформе .NET. Вильямс, 2003.
13. **Шалыто А. А.** Новая инициатива в программировании. Движение за открытую проектную документацию// Информационно-управляющие системы. 2003. № 4. С. 52–56.
14. **Thomas D., Barry B.** Model driven development – the case for domain oriented programming. http://www.oopsla.org/oopsla2003/files/ddd-session-vision.html
15. http: // www.omg.org / technology / documents / formal / xmi.htm
16. http: // www.omg.org / technology/ documents / formal / mof.htm
17. http://www.eclipse.org
18. **Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.** UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6. С. 12–17. http://is.ifmo.ru/works/UML-SWITCH-Eclipse.pdf
19. **Шалыто А. А.** SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. – 628 с. http://is.ifmo.ru/books/switch/3
20. **Шалыто А. А.** Автоматно-ориентированное программирование / Матер. IX Всероссийск. конф. по проблемам науки и высш. школы «Фундаментальные исследования в технических университетах». СПб.: Изд-во СПбГПУ, 2005. С. 44–52. http://is.ifmo.ru/works/_politeh.pdf
21. http://is.ifmo.ru/
22. **Гуров В. С., Мазин М. А., Шалыто А. А.** Операционная семантика UML-диаграмм состояний в программном пакете UniMod /Тр. XII Всероссийск. научно-методич. конф. «Телематика 2005». Т. 1. С. 74–76. http://tm.ifmo.ru/tm2005/src/224as.pdf

УДК 621.31

ПРОГРАММНЫЙ КОМПЛЕКС ОПРЕДЕЛЕНИЯ ПЕРЕГРУЗОК НА ЭТАПЕ КРАТКОСРОЧНОГО ПЛАНИРОВАНИЯ РЕЖИМА ЭКСПЛУАТАЦИИ СЕТИ

Н. Сингх,
канд. техн. наук, начальник отдела руководства проектами
Д. В. Чубраев,
канд. техн. наук, старший руководитель проекта
Компания «ETRANS AG» (Швейцария)

Одним из элементов системы обеспечения оперативной безопасности электропередающих сетей Западной Европы, входящих в союз UCTE (Union for the Co-ordination of Transmission of Electricity – Союз по координации передачи электроэнергии, объединяющий 34 системных оператора Европы), является процесс DACF (Day-Ahead Congestion Forecast – Прогноз перегрузок на день вперед), отвечающий за предсказание перегрузок на этапе краткосрочного планирования. В статье описывается подход к автоматизации процесса DACF на фирме ETRANS AG, являющейся координатором швейцарской передающей сети и энергетического блока «Юг» Европейской объединенной сети.

One of the important elements of operational security system of the European interconnected power network is the DACF (Day-Ahead Congestion Forecast) process, that is to be performed by all countries-members of UCTE (Union for the Co-ordination of Transmission of Electricity). This article describes the approach of Swiss Transmission System Coordinator ETRANS AG to the automation of the DACF process.

Введение

В условиях открытия энергетического рынка все более возрастает роль краткосрочного (на день вперед) планирования режима эксплуатации сети, что объясняется возрастанием доли краткосрочных сделок между поставщиками и потребителями энергии в общем объеме энергопоставок. Одновременно, ввиду повышения интенсивности межрегионального и межнационального энергообмена, доводится до максимума коэффициент использования сети, что означает, что сеть используется в режиме, наиболее близком к режиму максимально допустимой нагрузки. Таким образом, сочетание высокой нагруженности сети с краткосрочным изменением профиля производства и потребления энергии предъявляет ряд дополнительных требований как к процессу расчета и анализа режима работы сети, так и к соответствующим программным системам.

Ранее расчет режимов работы сети производился для небольшого количества контрольных случаев (например, зима–лето, день–ночь) и являлся частью процесса планирования сети, не ограниченного жесткими временными рамками, что позволяло осуществлять подобные операции группой специалистов с использованием интерактивных программных систем при большой доле ручных опреаций в процессе подготовки, расчета и анализа моделей сети. В настоящее время, помимо планирования, требуется краткосрочный расчет безопасности сети для ряда временных точек следующего дня, для целей оперативного управления сетью. Проведение подобного расчета традиционным «ручным» методом нецелесообразно, а, начиная с некоторого объема (для трех и более часов), и невозможно для одного человека, так как все вычисления должны быть завершены в течение одного-двух часов. Существующее положение дел выдвигает требование частичной или полной автоматизации процесса кратковременного расчета безопасности сети. С учетом использования современной программно-аппаратной базы, данная задача может быть успешно решена с помощью стандартных продуктов, доступных на рынке систем моделирования электрических сетей [1].

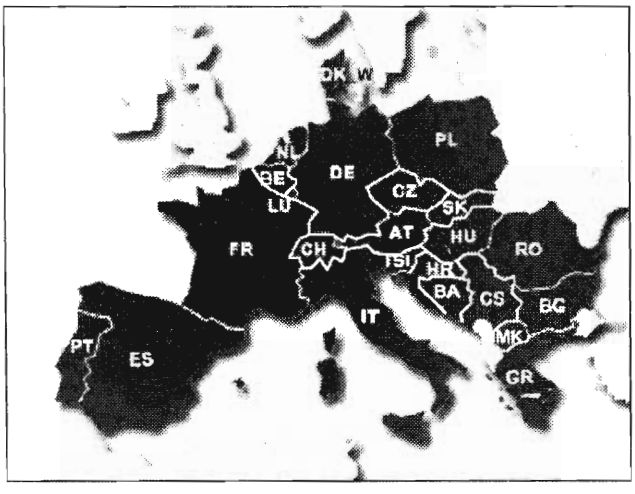
Подобное решение реализовано на фирме ETRANS, являющейся координатором швейцар-

ской передающей сети. Работа проведена в рамках проекта по разработке и реализации комплекса мер по предотвращению перегрузок высоковольтной передающей сети, основные принципы которого изложены в работе [2].

Постановка задачи

Система краткосрочного определения перегрузок в транспортной сети разработана в соответствии с указаниями UCTE (рис. 1) и основана на подходе, определенном рабочей группой DACF. Данный подход позволяет произвести построение и расчет объединенной сети, основываясь только на независимо созданных моделях частей этой сети. Согласно процессу, каждый из партнеров DACF готовит модель части сети, находящейся в его зоне ответственности, и помещает ее для общего пользования на сервер UCTE (Electronic Highway). В результате, каждая из сторон имеет доступ ко всем частичным моделям, соединив которые согласно алгоритму, описанному в документах UCTE [3] и статьях [2, 4, 5], получает модель всей сети, на базе которой могут производиться соответствующие вычисления, в первую очередь – расчет безопасности сети своей зоны ответственности (страны).

Несмотря на относительную простоту, процесс краткосрочного расчета безопасности сети по указанному алгоритму является достаточно трудоемким и занимает при ручной реализации до четырех рабочих часов на один случай расчета: порядка двух часов на построение модели своей части сети (взят случай сети размером 150 узлов) и столько же времени на проверку моделей партнеров, соединение их в модель общей сети, расчет безопасности и подготовку отчета. При увеличении количества случаев растут, соответственно, и затраты времени. При этом следует учесть, что построение частичных моделей невозможно до получения графиков производства и обмена энергией, становящихся доступными лишь в конце рабочего дня (16–17 ч), после заключения большей части сделок купли-продажи энергии, что дополнительно усложняет процесс ручной подготовки модели и расчета.



■ *Рис. 1. Страны-члены UCTE*

Чтобы высвободить высококвалифицированных специалистов, избавив их от рутинного труда, а также получить возможность предсказания состояния сети на каждый час следующего дня (в перспективе – на каждые 15 мин), на фирме ETRANS был начат проект по автоматизации процесса DACF. К настоящему времени реализационная часть проекта завершена и программная система в течение года находится в эксплуатации, что позволяет говорить в данной статье не только об ожиданиях от реализации системы, но и о связанных с опытом ее использования проблемах и шагах по ее усовершенствованию.

Требования, предъявляемые к системе

Практика показывает, что требования к современной системе оперативной безопасности сети, как и к большинству программных систем, подвержены постоянной модификации. Это объясняется рядом факторов, таких как связанное с открытием энергетического рынка изменение процессов в области электроэнергетики, изменение структуры фирм-производителей энергии, смена стандартов в области передачи информации.

Обсуждаемые программные комплексы активно используют численные методы решения систем дифференциальных уравнений для расчета установившихся режимов работы сети, при этом расчеты производятся для большого числа случаев в течение ограниченного времени (в особенности, в процессе оптимизационных расчетов), что выдвигает повышенные, по сравнению с системами планирования, требования к эффективности вычислений.

Таким образом, основная сложность состоит в необходимости сочетания системой высокой гибкости с высокой эффективностью и надежностью. В случае системы DACF фирмы ETRANS был выдвинут также ряд дополнительных требований:

– способность обмена информацией между различными операционными системами (OpenVMS, UNIX, MS Windows) так как под ними работают части систем управления, расчета, планирования сети, модули определения состояния сети (State Estimator) и др.;

– соответствие высоким требованиям безопасности и надежности;

– оптимальное использование персональных ресурсов: минимальное вовлечение персонала в работу системы в оперативном режиме; минимальные затраты на поддержание данных.

Программные системы в электротехнике, особенно задействованные в управлении сетью, имеют срок службы, измеряемый многими годами, иногда – десятками лет. За это время, как показывает практика, даже крупные поставщики систем успевают уйти с рынка, сменить профиль дея-

тельности или просто потерять ведущее положение в данной области. В настоящее время на фирме ETRANS имеется целый ряд систем, разработанных и установленных ведущими производителями мира, поддержание которых частично или полностью производится собственными сотрудниками фирмы, так как фирмы-поставщики больше не располагают квалифицированным персоналом для этой цели или вообще прекратили свое существование.

Перечисленные соображения приводят к требованию разработки системы, части которой, при необходимости, можно было бы заменить аналогичными продуктами другого производителя.

Требования надежности играют особенную роль в системе предсказания перегрузок, так как в этом случае речь идет о процессе, который, являясь критическим с точки зрения управления сетью, основывается на данных, качество которых может, в общем случае, варьироваться в очень широких пределах. Более того, необходимые для получения качественного результата данные могут отсутствовать на момент проведения расчета.

Все входящие данные, включая частичные модели сетей, должны проверяться на качество содержащейся в них информации. В случае прихода некачественных данных процесс должен найти им адекватную замену, гарантируя получение наиболее корректного в данной ситуации результата. При этом процессы проверки и подстановки данных должны удовлетворять требованиям надежности.

Требования к интерфейсу пользователя

Основной задачей системы DACF является представление результатов расчета для принятия решений и дальнейшего анализа. Данные должны быть представлены в виде, позволяющем оператору принять решение в течение ограниченного времени, что определяет требования к интерфейсу пользователя, который приближается к интерфейсу систем управления.

Наряду с результатами расчета должен контролироваться и сам процесс предсказания перегрузок, так как при своевременном обнаружении проблем с исходными данными могут быть предприняты меры для их корректировки.

Системные администраторы, отвечающие за надежность работы системы с точки зрения информатики, выдвигают свои требования к возможностям мониторинга состояния системы.

Помимо внутренних, результатами расчета пользуются и внешние пользователи: сотрудники зарубежных TSO (Transmission System Operator – Оператор передающей сети), представители операторов распределительных сетей, сотрудники департамента энергии и т. д. Для таких пользователей в защищенной области веб-страницы фирмы ETRANS было реализовано представление результатов $N/N-1$ расчета швейцарской сети.

Общий дизайн системы

Требования независимости системы от одного поставщика приводят к необходимости модульного дизайна. Данный подход также хорошо согласуется с необходимостью одновременной гибкости и эффективности системы.

Еще на этапе постановки задания было решено использовать основные модули только из числа стандартных продуктов, представленных на рынке, – как для того, чтобы иметь в распоряжении наиболее полно протестированные программы, так и для того, чтобы пользоваться усовершенствованиями, вносимыми в продукт в процессе развития.

Модульное построение системы налагает особые требования на межмодульные интерфейсы, так как именно они в дальнейшем будут определять возможности расширения системы и простоту замены ее отдельных частей.

Эти условия были учтены при дизайне системы, которая построена с использованием в качестве интерфейсов моделей сетей, имеющих четкое физическое значение:

– локальная сеть на данный момент времени;

– локальная сеть на аналогичный момент времени следующего дня (прогноз);

– общая модель европейской сети на следующий день.

В остальном система построена в соответствии с классической трехуровневой моделью, с разделением на данные, алгоритмы и интерфейс пользования.

Схема процесса с разбиением на программные модули приведена на рис. 2, описание алгоритмов работы системы – в работе [4].

Для реализации рассматриваемой системы требуются следующие компоненты:

– программа расчета сети со встроенным языком программирования для реализации алгоритмов пользователя;

– инструменты контроля процесса (workflows);

– системы хранения данных;

– системы обмена информацией (interprocess communication).

Правильный выбор базовых продуктов в существенной степени определяет надежность, удобство работы и, не в последнюю очередь, стоимость системы.

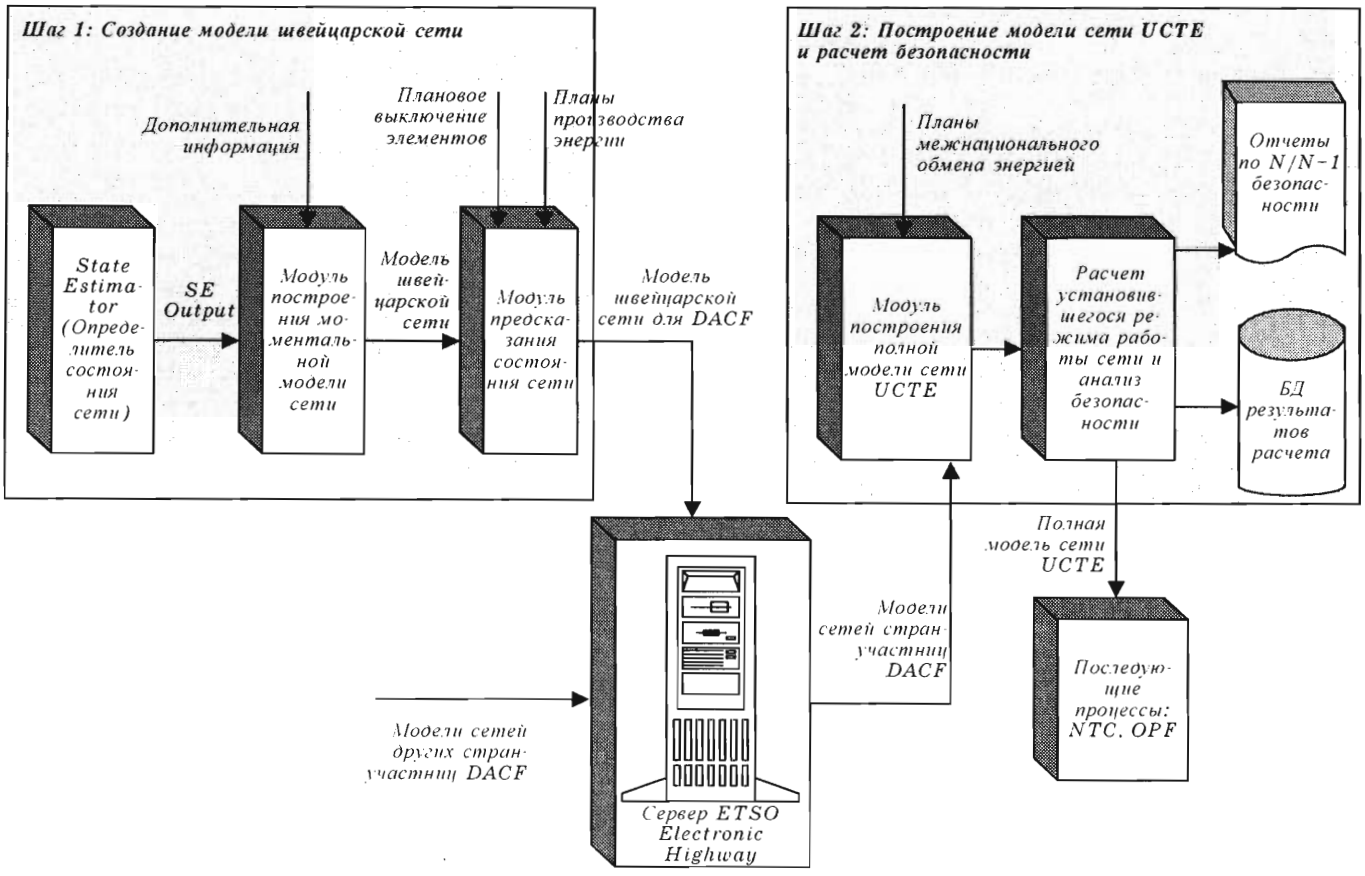
Требования к расчетной части системы

Основной задачей системы автоматического предсказания перегрузок является построение моделей сети, расчет $N/N-1$ безопасности сети и создание соответствующих отчетов. Используемый для этих целей продукт должен отвечать следующим требованиям:

– стабильность – так как все операции производятся в автоматическом режиме;

– гибкость – так как часть производимых операций не входит в набор стандартных функций программ расчета сетей;

– скорость – так как требуется произвести большое количество вычислений за ограниченное время;



■ *Рис. 2. Схема процесса DACF*

– развитость программного интерфейса – так как продукт должен быть интегрирован в систему workflows и обмениваться с ними информацией в обоих направлениях;

– развитость графического интерфейса пользователя – так как всегда может возникнуть необходимость дополнительного анализа системы в интерактивном режиме.

Одним из путей достижения поставленной задачи является разработка специального программного обеспечения. Однако, учитывая вопросы развитого графического интерфейса, поддержки и максимальной независимости от поставщика, от такого подхода было решено отказаться в самом начале проекта.

Соответственно, первым шагом в создании системы был анализ доступных на рынке программных пакетов для расчета сетей.

Одним из важнейших критериев отбора являлось быстродействие расчетного ядра системы, однако вопросы надежности, сходимости численных алгоритмов, гибкости, интерфейса пользователя, точности моделирования элементов сети, возможности дальнейших расширений также играли существенную роль. Для выбора системы, наиболее подходящей для решения поставленной задачи, было отобрано 6 продуктов ведущих разработчиков Европы и США и проведен ряд тестов, из результатов которых следовало, что ни один из рассматриваемых продуктов не смог стать лидером во всех категориях, а по некоторым категориям разброс показателей оказался очень велик. Так, например, расчет $N-1$ безопасности самым быстрым кандидатом проводился за 20 с, самым медленным – за 10 мин, т. е. разница в быстродействии достигала 30 раз (расчет $N/N-1$ безопасности для 200 элементов сети из 2500 узлов). Кроме того, некоторые продукты не обеспечивали требуемую надежность работы, многие не соответствовали требованиям гибкости, интеграции в процесс и автоматизации операций.

Особенные требования предъявлялись и к возможностям моделирования, так как в швейцарской сети и сетях соседних стран используются трехобмоточные трансформаторы с угловым регулированием, не моделировавшиеся с достаточной точностью большинством программных пакетов ввиду отсутствия подобных устройств в сетях США и большинства европейских стран.

В результате интенсивного тестирования был определен продукт, отвечавший на момент рассмотрения всем основным требованиям. Недостатком продукта являлось несовершенство графического интерфейса пользователя. Данное требование, од-

нако, не было основным при реализации автоматизированной системы, а фирма-разработчик объявила о запланированном выпуске новой версии пакета (на данный момент поступившей в продажу), где этот недостаток был устранен. В то же время, продукт показал высокую скорость расчета, хорошую сходимость алгоритмов, развитые возможности моделирования, а также очень высокую гибкость благодаря встроенному языку автоматизации.

Workflows – это программы, работающие в событийно-ориентированном режиме и перенимающие ответственность за правильную последовательность операций во время процесса. Они относятся к так называемым продуктам класса «middleware» – «связывающие, обеспечивающие взаимодействие». Выбор программ workflows достаточно широк, но количество программ для автоматизации инженерных процессов очень ограничено. Так как workflow всегда отражает сугубо индивидуальный процесс пользователя, данную часть системы можно реализовать на любом из языков программирования: функции открытия и копирования файлов, чтения файлов в различных форматах, работа по событию (и, как частный случай, по таймеру) поддерживаются всеми языками высокого уровня под MS Windows. Но требования к системе workflow идут существенно дальше, включая возможности диагностики, тестирования, удаленной отладки и удаленного запуска, ведения журналов работы и многое другое. Разработка подобной функциональности с нуля неоправданна и ведет к существенному повышению стоимости системы при увеличении сроков ее реализации.

На фирме уже была установлена система workflows, которая и была использована при реализации данного программного комплекса.

Рассматриваемая система оперирует большим количеством данных, объем которых может доходить до гигабайта в день. Большую часть этих данных составляют модели сетей Швейцарии и стран-участниц UCTE, хранящиеся в файлах формата UCTE и стандартном для отрасли формате RAW (оба – текстовые форматы представления модели сети). Планы производства и обмена энергией хранятся в виде файлов формата CSV (Comma-Separated Values – данные, разделенные запятой), достаточно удобном для машинного и операторского чтения. Ввиду сравнительно небольшого объема этих данных, было решено воздержаться от использования базы данных в пользу простоты их обмена, контроля и ручной модификации. Все данные хранятся в системе директорий, структура которых позволяет быстрый автоматизированный и ручной доступ к информации, а также ее удобную архивацию.

Система предсказания перегрузок сети представляет собой набор модулей, согласованная работа которых невозможна без обмена информацией с целью передачи параметров запуска и возврата кода завершения процесса. Для решения данной задачи требуется простая и эффективная система межпроцессной коммуникации.

В рассматриваемой системе используется метод обмена информацией на базе протокола HTTP. Использование метода GET протокола HTTP позволяет осуществить межпроцессную межплатформенную коммуникацию для всех вовлеченных систем, без применения флаг-файлов, часто используемых для синхронизации различных процессов.

Workflows

Процесс расчета безопасности сети основан на разработанном рабочей группой DACF алгоритме расчета безопасности сети, являющемся на настоящий момент стандартом в рамках UCTE. По описанному алгоритму производится расчет практически во всех странах-членах UCTE, при этом все операции или их часть проводятся вручную.

При постановке задания на разработку рассматриваемой системы было указано на важность создания полностью автоматизированного процесса, позволяющего производить полный цикл расчета без непосредственного участия в нем специалистов.

Это достаточно сложная задача, так как в процессе расчета используются данные, поставляемые из различных систем и имеющие различное качество, в частности – модели национальных передающих сетей 22 стран.

Обычно оценка, корректировка или эквивалентная замена моделей проводится опытным инженером-электротехником. В описываемой системе эту функцию берут на себя элементы программного комплекса. Реализация подобной функциональности требует гибкого и мощного инструмента, который производит поиск и скачивание модели с сервера; проверку данных на формальное соответствие стандартам; формальный анализ модели сети, ее расчет с последующим анализом результатов и сравнением их с предопределенным набором критических значений; замену моделей, не прошедших проверку, эквивалентными моделями, измененными таким образом, чтобы они максимально соответствовали моделируемому моменту времени. После ряда тестов было принято следующее разделение функциональности:

– все операции, связанные с проверкой, моделированием, расчетом и анализом сети, реализовать в модулях на базе системы расчета сети;

– все операции, связанные с поиском, проверкой, передачей данных, а также контроль процессов во времени (запуск по таймеру и по событию), возложить на систему контроля workflows.

В рассматриваемой системе workflows, являясь контрольным механизмом, также реализуют и функции оповещения персонала в случае ошибок. Очевидно, что стабильность workflows – основной определяющий фактор стабильности процесса в целом.

Опыт эксплуатации системы

Описываемая система находится в действии более года. Проведенные за это время тесты показали хорошее соответствие результатов, полученных при автоматическом процессе, с результатами, основывающимися на ручной подготовке моделей, что не является, как может показаться, само собой разумеющимся, так как, несмотря на достаточно хорошую алгоритмизацию процесса в целом, практически при каждом расчете приходится сталкиваться с рядом проблем, которые относительно легко могут быть решены специалистом, но представляют сложности для реализации в автоматизированной программной системе.

Проблема первая: недопоставленные данные. Все данные в рассматриваемую систему поставляются из других систем, частично – в результате автоматизированного, частично – в результате ручного процесса. Оба могут давать сбои, приводящие к непоставке, несвоевременной поставке или искажению данных. Каким образом решается проблема? Все данные делятся на несколько групп по признаку их важности для рассматриваемого процесса.

- Критичные – при их отсутствии процесс прерывается. После длительных тестов к группе критичных данных было решено относить только модель локальной (в данном случае – швейцарской) сети, так как при ее отсутствии результат может настолько отличаться от планируемого, что пользоваться им будет нецелесообразно.
- Важные – при их отсутствии система может самостоятельно создавать подходящую замену (одновременно информируя пользователей, что результат нуждается в дополнительной проверке) или запросить помощи оператора. К таким данным относятся, например, модели сетей, окружающих анализируемую часть сети (в данном случае – модели пограничных со Швейцарией стран – Германии, Италии, Франции и Австрии).
- Второстепенные – при их отсутствии система также ищет или создает подходящую замену, но специальное предупреждение не выдается, так как влияние их на результат обычно невелико. В качестве примера можно привести модели сетей удаленных стран, планы производства энергии небольших электростанций.

Как показывает практика, по ряду причин (как технических, так и организационных) регулярный своевременный приход полного набора данных от всех участников процесса на данном этапе невозможен, что ведет к необходимости реализации алгоритмов подстановки для всех важных и второстепенных типов данных – планов производства и обмена энергией, моделей сетей и др.

Проблема вторая: данные, содержащие ошибки. Хотя в ряде случаев ошибки в данных определить практически невозможно (например, неверное значение в плане производства энергии, не выходящее за физические границы станции), в большинстве случаев подобная возможность существует. Наиболее важной является проверка на корректность моделей сетей стран, при этом проверяется как формат данных, так и модель в целом. В системе DACF реализована многоуровневая проверка данных, включающая проверку формата с одновременной конвертацией модели в расчетный формат; проверка соответствия генерации в узлах сети заданным физическим лимитам; проверка баланса активной и реактивной мощности, генерации и нагрузки и, наконец, численный расчет модели. Модели, не прошедшие проверку, заменяются по тому же алгоритму, что и отсутствующие данные.

Проблема третья: невозможность найти численное решение для режима работы сети (отсутствие сходимости алгоритма). Подобное явление представляет неотъемлемый риск при использовании численных методов расчета, хотя его вероятность и может быть существенно снижена предварительной проверкой частичных моделей. Специалист, в случае отсутствия сходимости, пробует различные алгоритмы расчета, одновременно снимая наложенные ограничения, или модифицирует модель с целью устранения причины проблемы. Реализация подобного подхода в автоматизированной системе может быть затруднена, в частности, из-за временных ограничений, налагаемых процессом.

На основе результатов проведенного анализа было решено остановиться на подходе, максимально приближенном к последовательности операций, проводимой специалистом при ручном расчете:

– расчет методом Ньютона–Рапсона (Decoupled Newton–Rapsohn) с учетом лимитов на реактивную мощность. Данный метод по результатам тестов обладает наилучшим соотношением сходимость–скорость;

– в случае отсутствия сходимости повторяется расчет методом Ньютона без учета лимитов на реактивную мощность и с использованием плавного старта (flat start);

– в случае повторного отсутствия сходимости используется модифицированный метод Гаусса без учета лимитов на реактивную мощность.

Таким образом, по своим возможностям система DACF близка к тому, что может обеспечить квалифицированный специалист, выполняющий ручной расчет по данной методике. При этом построение модели европейской сети (модели 22 стран, общий размер сети 2500 узлов) и расчет на ее базе безопасности швейцарской сети ($N/N-1$ для 250 элементов) занимает около 2,5 мин, что позволяет произвести расчет для 24 ч следующего дня примерно за 1 ч. Эффективное время расчета составляет при этом менее 20 %, остальные 80 % занимает проверка данных и подготовка отчетов. Как видно из рис. 3, это принципиальное ускоре-

ние по сравнению с ручным решением, занимающим от двух часов и более.

Опыт, собранный в процессе начальной эксплуатации системы, оказал влияние на дальнейший подход к ее дизайну. Вот некоторые из выводов.

1. Становление полностью автоматизированного решения является процессом, требующим определенного времени (возможно, более года), в течение которого проявляют себя все непредусмотренные комбинации факторов, влияющие на результат. В течение этого периода необходимо считаться с доработками в алгоритмах анализа данных, параметрах расчета, workflows и др.

2. Какой бы надежной не казалась реализация автоматического выполнения последовательности операций (workflow), всегда должна быть предусмотрена возможность ручного запуска всех операций последовательности. Данная функциональность важна в случае непредвиденного поведения системы, выхода из строя системы workflow или отказа части аппаратного обеспечения системы.

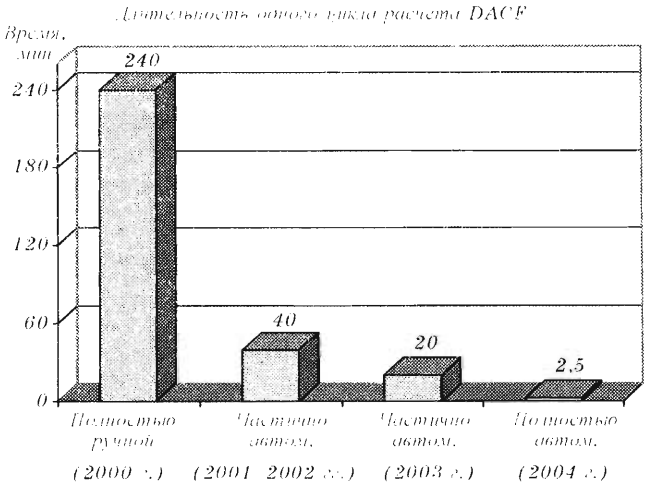
3. Если целью является построение полностью автоматической системы, разумно так организовать процесс, чтобы результат выдавался даже при минимальном наборе исходных данных. При этом целесообразно ввести коэффициент качества, изменяющегося, например, с 10 % (минимум достоверных данных) до 100 % (наличие всех необходимых данных).

4. Если целью является получение наиболее высокого качества результатов, необходимо предусмотреть возможность корректировки оператором как частичных данных, так и времени выполнения действий процесса (например, имеет смысл сдвинуть время запуска расчета, если известно, что необходимые данные задерживаются, но будут доступны в течение допустимого времени).

Как уже было сказано выше, система DACF постоянно развивается и совершенствуется. В качестве следующих шагов предусматривается реализация интерфейса OPC (OLE for Process Control) для упрощения мониторинга процесса и связи с системой управления сетью, улучшение системы диагностики и дальнейшее развитие системы архивации и представления результатов расчета.

Заключение

Система предотвращения перегрузок является необходимым инструментом каждого независимого оператора передающей сети. Освещенный в данной статье программный комплекс предотвращения перегрузок на этапе краткосрочного планирования является важным шагом на пути автоматизации процесса оперативного управления сетью в условиях рынка. Помимо обеспечения прогноза режима работы сети на основании наиболее реалистичного на сегодняшний день для европейской сети метода DACF, комплекс поставляет модели



■ *Рис. 3. Длительность одного полного цикла расчета по методу DACF*

для целого ряда других систем: расчета пропускной способности сети, расчета мер по устранению перегрузок [7], оперативного расчета безопасности сети.

Рассмотренный программный комплекс вызвал большой интерес представителей стран Южной и Восточной Европы на конференции MedPower 2004 и, с точки зрения авторов, может представлять интерес и для операторов российской передающей сети, особенно в условиях совместного проекта по исследованию синхронного объединения энергосистем Западной, Центральной и Южной Европы с ЕЭС России.

Литература

1. **Dy-Liacco T., Singh N., Pavella M.** Congestion Management in a Large Interconnected Network: Needs and Methods// IFAC. Shanghai, 2003.
2. **Zimmerman D., Imhof K., Emery M.** Modular Day-ahead Congestion Forecast as a First Step of a Congestion Management Process: Proc. of the 1st Balkan Power Conf. Bled, Slovenia. 2001.
3. UCTE Operational Handbook, Policy 4. www.ucte.org
4. **Tchoubraev D., Singh N., Chan K. H.** et al. Advanced Automated Approach for Interconnected Power System Congestion Forecast: Proc. of conf. MedPower 2004. Cyprus, 2004.
5. **Emery M., Zeitler G.** Tagesgerechte Engpassvorhersage (DACF): VDE-Kongress. Berlin, 2004.
6. **Singh N., Tchoubraev D.** Operational Security Analysis of Interconnected European Network in Liberalized Market: Conf. PowerTech 2005. St.Petersburg, 2005.
7. **Emery M., Karpatchev A., Tchoubraev D.** Congestion Management at ETRANS: 2nd CIGRE / IEEE PES International Symposium «Congestion Management in a Market Environment», San Antonio, USA, 2005.

УДК 681.3.06:62-507

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

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. Приводится пример решения физической задачи, 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.

Реализация игры «Жизнь»

Клеточный автомат Джона Хортона Конвея «Жизнь» [1–3] настолько известен, что нет необходимости приводить словесное описание его правил. Данный раздел посвящен пяти вариантам реализации этого автомата: простейшей реализации «в лоб», реализации с зональной оптимизацией, реализации для многопроцессорной и кластерной вычислительных систем, для обобщенных координат [4].

Простейшая реализация. Приведем реализацию игры «Жизнь» для однопроцессорной системы, без оптимизации, для привычной картезианской метрики.

При этом необходимо разработать класс, реализующий соответствующий компонент правил. Назовем его `CALifeRules`. В качестве решетки автомата можно выбрать любую из стандартных двумерных решеток (наиболее привычна квадратная, стандартный компонент «Square Basic Grid»). В качестве метрики – картезианские координаты (стандартный компонент «Cartesians»), а хранилища данных – хранилище для булевых величин (стандартный компонент «Booleans for Cartesians»).

Необходимо отметить, что перед проведением эксперимента потребуется включить хранение данных на двух последовательных шагах (параметр «Double» хранилища перевести в значение `true`). В результате внутри компонента будут созданы две копии структуры данных для размещения состояния решетки. Значения будут получаться из одной из них, а помещаться – в другую. После каждого шага структуры будут меняться местами. Это обеспечит независимость результирующего состояния решетки от порядка перебора клеток – неотъемлемое свойство «классических» клеточных автоматов.

Введем в разрабатываемом компоненте правил два набора по 13 параметров булевого типа (массивы `p_bBorn` и `p_bDie`), которые позволят определять функцию переходов автомата. Если значение параметра `p_bBorn[i]` равно `true`, то мертвая клетка оживает при наличии *i* живых соседей. Если значение параметра `p_bDie[i]` равно `true`, то живая клетка умирает при наличии *i* живых соседей. Значение *i* лежит от нуля до двенадцати (наибольшего числа непосредственных соседей на треугольной решетке), однако на квадратной решетке будут использованы лишь первые девять из них (от нуля до восьми).

Кроме того, введем целочисленный анализируемый параметр p_iAlive, который позволит наблюдать за динамикой изменения количества живых клеток на решетке.

Для сопоставления компоненту некой иконки необходимо создать соответствующий ресурс. Пусть его идентификатор будет IDI_ICON.

Объявление класса CALifeRules может иметь следующий вид:

```
class CALifeRules:public CARules
{
public:
    // Конструктор и деструктор
    CALifeRules();
    virtual ~CALifeRules();

    // Определение имени, описания, иконки и условий совместимости
    COMPONENT_NAME(Game of Life (plain))
    COMPONENT_INFO(Conways game of Life rules (plain variant))
    COMPONENT_ICON(IDI_ICON)
    COMPONENT_REQUIRES(Data.bool.*&Metrics.2D.cartesian.*)

    // Следующие функций класса CARules будут переопределены
    virtual bool Initialize();
    virtual bool SubCompute(Zone& z);

public:
    // Параметры
    ParamBool* p_bBorn[13];
    ParamBool* p_bDie[13];

    // Анализируемый параметр
    ParamInt p_iAlive;

public:
    // Карта параметров
    PARAMETERS_COUNT(26)
    BEGIN_PARAMETERS
        PARAMETER(0,p_bBorn[0])
        PARAMETER(1,p_bBorn[1])
        PARAMETER(2,p_bBorn[2])
        PARAMETER(3,p_bBorn[3])
        PARAMETER(4,p_bBorn[4])
        PARAMETER(5,p_bBorn[5])
        PARAMETER(6,p_bBorn[6])
        PARAMETER(7,p_bBorn[7])
        PARAMETER(8,p_bBorn[8])
        PARAMETER(9,p_bBorn[9])
        PARAMETER(10,p_bBorn[10])
        PARAMETER(11,p_bBorn[11])
        PARAMETER(12,p_bBorn[12])
        PARAMETER(13,p_bDie[0])
        PARAMETER(14,p_bDie[1])
        PARAMETER(15,p_bDie[2])
        PARAMETER(16,p_bDie[3])
        PARAMETER(17,p_bDie[4])
        PARAMETER(18,p_bDie[5])
        PARAMETER(19,p_bDie[6])
        PARAMETER(20,p_bDie[7])
        PARAMETER(21,p_bDie[8])
        PARAMETER(22,p_bDie[9])
        PARAMETER(23,p_bDie[10])
        PARAMETER(24,p_bDie[11])
        PARAMETER(25,p_bDie[12])
    END_PARAMETERS

    // Карта анализируемых параметров
    APARAMETERS_COUNT(1)
    BEGIN_APARAMETERS
        PARAMETER(0,&p_iAlive)
    END_APARAMETERS

private:
    // Вспомогательная переменная для вычисления значения p_iAlive
    int iq;
};
```

Три необходимых каждой библиотеке компонента функции можно создать с помощью описанных выше макроопределений:

```
COMPATIBLE_RULES(1.0)
RULES_COMPONENT(CALifeRules)
```

Основное назначение конструктора и деструктора компонента состоит в создании, инициализации, а также удалении параметров. В данном случае они будут иметь вид

```
CALifeRules::CALifeRules():
      p_iAlive("Alive","Amount of alive cells",0)
{
      p_bBorn[0]=new ParamBool("Born if 0","Is dead cell to become alive if it has 0 alive neighbors",false,this);
      p_bBorn[1]=new ParamBool("Born if 1","Is dead cell to become alive if it has 1 alive neighbors",false,this);
      p_bBorn[2]=new ParamBool("Born if 2","Is dead cell to become alive if it has 2 alive neighbors",false,this);
      p_bBorn[3]=new ParamBool("Born if 3","Is dead cell to become alive if it has 3 alive neighbors",true,this);
      p_bBorn[4]=new ParamBool("Born if 4","Is dead cell to become alive if it has 4 alive neighbors",false,this);
      p_bBorn[5]=new ParamBool("Born if 5","Is dead cell to become alive if it has 5 alive neighbors",false,this);
      p_bBorn[6]=new ParamBool("Born if 6","Is dead cell to become alive if it has 6 alive neighbors",false,this);
      p_bBorn[7]=new ParamBool("Born if 7","Is dead cell to become alive if it has 7 alive neighbors",false,this);
      p_bBorn[8]=new ParamBool("Born if 8","Is dead cell to become alive if it has 8 alive neighbors",false,this);
      p_bBorn[9]=new ParamBool("Born if 9","Is dead cell to become alive if it has 9 alive neighbors",false,this);
      p_bBorn[10]=new ParamBool("Born if 10","Is dead cell to become alive if it has 10 alive neighbors",false,this);
      p_bBorn[11]=new ParamBool("Born if 11","Is dead cell to become alive if it has 11 alive neighbors",false,this);
      p_bBorn[12]=new ParamBool("Born if 12","Is dead cell to become alive if it has 12 alive neighbors",false,this);

      p_bDie[0]=new ParamBool("Die if 0","Is alive cell to become dead if it has 0 alive neighbors",true,this);
      p_bDie[1]=new ParamBool("Die if 1","Is alive cell to become dead if it has 1 alive neighbors",true,this);
      p_bDie[2]=new ParamBool("Die if 2","Is alive cell to become dead if it has 2 alive neighbors",false,this);
      p_bDie[3]=new ParamBool("Die if 3","Is alive cell to become dead if it has 3 alive neighbors",false,this);
      p_bDie[4]=new ParamBool("Die if 4","Is alive cell to become dead if it has 4 alive neighbors",true,this);
      p_bDie[5]=new ParamBool("Die if 5","Is alive cell to become dead if it has 5 alive neighbors",true,this);
      p_bDie[6]=new ParamBool("Die if 6","Is alive cell to become dead if it has 6 alive neighbors",true,this);
      p_bDie[7]=new ParamBool("Die if 7","Is alive cell to become dead if it has 7 alive neighbors",true,this);
      p_bDie[8]=new ParamBool("Die if 8","Is alive cell to become dead if it has 8 alive neighbors",true,this);
      p_bDie[9]=new ParamBool("Die if 9","Is alive cell to become dead if it has 9 alive neighbors",true,this);
      p_bDie[10]=new ParamBool("Die if 10","Is alive cell to become dead if it has 10 alive neighbors",true,this);
      p_bDie[11]=new ParamBool("Die if 11","Is alive cell to become dead if it has 11 alive neighbors",true,this);
      p_bDie[12]=new ParamBool("Die if 12","Is alive cell to become dead if it has 12 alive neighbors",true,this);
```

```
      // Определение комментария, отображаемого в строке состояния
      sComment[1]="Game of Life";
}

CALifeRules::~CALifeRules()
{
      for(unsigned char b=0;b<13;b++) delete p_bBorn[b];
      for(b=0;b<13;b++) delete p_bDie[b];
}
```

Потребность в обработчике инициализации (перегрузке функции `Initialize`) возникает только из-за наличия анализируемого параметра, значение которого должно быть известно перед выполнением первой итерации. В результате эта функция примет вид

```
bool  CALifeRules::Initialize()
{
    // Инициализацию следует производить, только если
    //параметр анализируется
    if (!p_iAlive.IsAnalyzed()[2]) return true;

    DATUM[3](CACrtsBoolDatum);
    CACell c; // Текущая клетка
    iq=0;

    for(int i=(int)datum->z.a1; i<=(int)datum->z.b1; i++)
         for(int j=(int)datum->z.a2; j<=(int)datum->z.b2; j++)
       {
              // Получение идентификатора текущей клетки
              // по декартовым координатам
               c=GetUnion()->Metrics->ToCell(i,j,0);

              if (datum->Get(c)) iq++;
       }
    // Присваивание значения анализируемому параметру
    p_iAlive.Set(iq);

    return true;
}
```

Необходимо отметить, что переменная-член `Zone z` класса `CADatum` описывает зону, значения состояний клеток из которой содержатся в хранилище. Эта переменная используется в приведенной выше функции для организации перебора всех клеток в цикле.

Далее приводится основная функция, осуществляющая итерации. Как отмечалось выше, даже для однопроцессорной системы, не предусматривающей распараллеливания вычислений, реализацию шага лучше осуществлять в функции `SubCompute`. Пример такого решения:

```
bool  CALifeRules::SubCompute(Zone&  z)
{
    DATUM(CACrtsBoolDatum);

    CACell c; // Текущая клетка
    CACell neig[12]; // Массив для хранения всех соседей
    // Переменная, хранящая количество соседей
    unsigned short ncount=GET_METRICS[4]->GetNeighboursCount();
    unsigned char alive; // Количество живых соседей
                         // текущей клетки
    iq=0;

    for(CACell i=z.a1; i<=z.b1; i++)
         for(CACell j=z.a2; j<=z.b2; j++)
       {
            alive=0;

              // Получение идентификатора текущей клетки по декартовым
              // координатам
               c=GetUnion()->Metrics->ToCell(i,j,0);

              // Получение всех соседей и размещение их в массиве
               pUnion->Metrics->GetNeighbours(c,neig);

              // Подсчет числа живых соседей текущей клетки
              for (int k=0; k<ncount; k++) {
                    if (datum->Get(neig[k])) alive++;
             }

              // Применение функции переходов и определение значения
              // анализируемого параметра
             if (datum->Get(c)) {
                    datum->Set(c,!p_bDie[alive]->Get());
                   if (!p_bDie[alive]->Get()) iq++;
              } else {
                    datum->Set(c,p_bBorn[alive]->Get());
                   if (p_bBorn[alive]->Get()) iq++;
             }
       }
    // Присваивание значения анализируемому параметру
    p_iAlive.Set(iq);

    return true;
}
```

Таким образом, пример простейшей реализации клеточного автомата «Жизнь» построен. Однако следует отметить, что перед его использованием компонент необходимо скомпилировать и уставить в среду, для чего требуется выбрать в главном меню «Tools» | «Components Manager...» (или воспользоваться горячей клавишей <F9>), затем нажать в появившемся окне кнопку «Add...» и открыть файл библиотеки компонента.

В настоящей работе не будут обсуждаться такие вопросы, как включение необходимых заголовочных файлов и подключение библиотек, в силу громоздкости описания этой тематики, а также для того, чтобы не обижать читателей предположением, что они не смогут сделать этого самостоятельно.

В заключение отметим, что вычисление анализируемых параметров в процессе выполнения шага автомата позволяет существенно сэкономить вычислительное время и избавиться от повторных переборов всех клеток решетки.

Реализация с использованием зональной оптимизации. Понаблюдав за моделированием рассматриваемого клеточного автомата, нетрудно заметить, что почти всегда изменения состояний клеток про-

[1] Переменная `sComment` – член класса `CARules`.

[2] Функция `IsAnalyzed()` – член класса `Parameter`.

[3] Данное макроопределение создает локальную переменную `datum`, указатель на хранилище данных, с которым работает компонент правил. При этом он приводится к типу указателя на класс, передаваемый макроопределению в качестве параметра.

[4] Макроопределение `GET_METRICS` обеспечивает быстрый доступ к метрике, с которой работает данный компонент. Аналогично имеются макроопределения `GET_GRID` и `GET_DATUM`.

исходят лишь в некоторых локализованных областях. При этом большие зоны, состоящие сплошь из мертвых клеток, можно исключать из рассмотрения, так как на данном шаге жизнь в них зародиться не сможет. Отбрасывание таких областей будем называть зональной оптимизацией вычислений.

Для осуществления такой оптимизации в библиотеке CADLib имеется класс `OptZonal`. На каждой итерации объект этого класса запоминает клетки, состояние которых может повлиять на соседей на следующих итерациях (назовем их значимыми клетками). Пользуясь этой информацией, он формирует зоны, которые следует принимать в расчет при вычислениях на следующем шаге.

Приведем описание нескольких членов класса, которые будут использованы для реализации клеточного автомата «Жизнь» с зональной оптимизацией.

- `public OptZonal(unsigned int diffusion,unsigned int unite)` – конструктор. Параметр `diffusion` определяет величину «полей» зон, расстояние от крайних значимых клеток до границы зоны. Фактически, этот параметр должен быть равен радиусу окрестности клеток [4]. Параметр `unite` определяет наибольшее расстояние между клетками двух зон, при котором их следует объединять в одну.
- `public inline void Reset()` – функция инициализирует объект.
- `public void AddCell(CACell x,CACell y=0,CACell z=0)` – функция добавляет информацию о значимой клетке с соответствующими координатами вдоль трех осей.
- `public void NewStep()` – функция переключает указатель на рабочее хранилище информации о зонах между двумя внутренними хранилищами, проверяет, перекрываются ли зоны в них, и т. п. Она должна вызываться перед каждым новым шагом для того, чтобы объект данного класса вычислял границы зон для следующего шага.
- `public bool ReleaseZone(Zone& z)` – функция возвращает через параметр `z` границы зоны, которую следует обработать. Ее следует вызывать до тех пор, пока результат, возвращаемый самой функцией, не станет равным `false`.

Для того чтобы воспользоваться зональной оптимизацией, в решение, разработанное в подразд. «Простейшая реализация» (с. 30), необходимо внести ряд изменений.

Во-первых, потребуется добавить переменную – член класса, реализующего компонент правил, `OptZonal oOpt`. В конструкторе класса `CALifeRules` необходимо вызвать конструктор для этой переменной с соответствующими параметрами, например `oOpt(1,4)`, а также присвоить значение `true` переменной `bFinalizeIfChanged`[1]. В результате изменение состояние решетки вручную приведет к финалу эксперимента. Это необходимо для обеспечения целостности информации о зонах.

Во-вторых, в функции `Initialize` класса, реализующего правила, для обеспечения корректности информации об обновляемых зонах необходимо сделать обнуление второго буфера данных, если включено двойное хранилище. В результате функция примет вид

```
bool  CALifeRules::Initialize()
{
     // Инициализация оптимизатора
     oOpt.Reset();

     DATUM(CACrtsBoolDatum);
     CACell c; // Текущая клетка
     iq=0;

     for(CACell i=datum->z.a1; i<=datum->z.b1; i++)
          for(CACell j=datum->z.a2; j<=datum->z.b2; j++)
        {
               // Получение идентификатора текущей клетки
               // по декартовым координатам
                c=GetUnion()->Metrics->ToCell(i,j,0);

              if (datum->Get(c)) {
                iq++;
                   // Данная клетка живая, а следовательно -
                   // значимая
                  oOpt.AddCell(i,j);
            }
        }
     // Присвоение значения анализируемому параметру
     p_iAlive.Set(iq);

     // Обнуление второе буфера данных, если включено
     // двойное хранение
     if (datum->p_bDouble.Get()) {
          for(CACell i=datum->z.a1; i<=datum->z.b1; i++)
               for(CACell j=datum->z.a2; j<=datum->z.b2; j++)
           {
                    c=GetUnion()->Metrics->ToCell(i,j,0);
                  datum->Set(c,false);
           }
    }

     return true;
}
```

В-третьих, в функции `SubCompute` необходимо добавить вызов функции `oOpt.NewStep` и производить вычисления только для тех зон, которые возвращаются функцией `oOpt.ReleaseZone`. Кроме этого, как и в обработчике инициализации, необходимо вызывать функцию `oOpt.AddCell` для всех оживших клеток.

В результате этих изменений производительность вычислений повысится, в среднем, на порядок.

Реализация для многопроцессорной системы. Идея выполнения клеточного автомата «Жизнь» на многопроцессорной вычислительной системе заключается в создании большого числа потоков [5], каждый из которых будет выполнять функцию `SubCompute` для некоторой части пространства. Потоки, в свою очередь, распределяются между процессорами. Итерация

[1] Переменная `bFinalizeIfChanged` – член класса `CARules`.

считается законченной, когда завершились вычисления во всех потоках.

Для того чтобы рассматриваемый клеточный автомат выполнялся на многопроцессорной системе, в решение, разработанное в подразд. «Простейшая реализация», необходимо внести ряд изменений.

Во-первых, удалить из функции SubCompute обнуление временной переменной iq, а также присвоить нового значение анализируемому параметру p_iAlive.

Во-вторых, реализовать функцию Compute следующим образом (все используемые в данном фрагменте кода макроопределения реализованы в библиотеке CADLib):

```
bool  CALifeRules::Compute()
{
      DATUM(CACrtsBoolDatum);

      // Создание счетчиков стартовавших и завершившихся
      // потоков
      COUNTER(started);
      COUNTER(finished);

      // Инициализация будущего значения анализируемого параметра
     iq=0;

      // Разделение задачи на подзадачи для разных потоков
      MULTIFOR2(k1,4*(int)pNetwork->GetThis()->uCPUsCount[1],
            datum->z.a1,datum->z.b1,
            k2,4*(int)pNetwork->GetThis()->uCPUsCount,
            datum->z.a2,datum->z.b2,started,finished);
      // Возможен и такой вариант:
      //  MULTIFOR2_FIXED(k1,100,datum->z.a1,datum->z.b1,
      //        k2,100,datum->z.a2,datum->z.b2,started,finished);

      // Ожидание совпадения счетчиков. Проверка каждые
      // 20 миллисекунд
      WAIT_COUNTER(started,finished,20);

      // Присвоение значения анализируемому параметру
      p_iAlive.Set(iq);

      return true;
}
```

Комментарии делают ясным назначение каждой строки приведенной выше функции. Отдельно рассмотреть имеет смысл только макроопределения MULTIFOR2 и MULTIFOR2_FIXED. Запишем их в общем виде.

Макроопределение MULTIFORn(k1, count1, a1, b1,..., kn, countn, an, bn, started, finished) осуществляет деление n-мерной задачи (n лежит от одного до трех) на подзадачи, причем отрезок [ai, bi] вдоль i-й оси делится на counti частей. Параметр ki определяет имя создаваемой локальной переменной цикла вдоль i-й оси (i лежит от одного до n). Таким образом, для $n = 3$ будет создано count1*count2*count3 потоков. Последние два параметра передают названия счетчиков стартовавших и завершившихся потоков соответственно. Использование двух различных счетчиков (вместо одного) оправдывается рядом весомых соображений, обсуждение которых выходит за рамки настоящей работы.

Макроопределение MULTIFORn_FIXED(k1, size1, a1, b1,..., kn, sizen, an, bn, started, finished) осуществляет деление n-мерной задачи (n лежит от одного до трех) на подзадачи, причем отрезок [ai, bi] вдоль i-й оси делится на отрезки по sizei клеток (последний из них может быть меньше). Параметр ki определяет имя создаваемой локальной переменной цикла вдоль i-й оси (i лежит от одного до n). Таким образом, для $n = 3$ будет создано ((b1-a1+1)/size1)*((b2-a2+1)/size2)*((b3-a3+1)/size3) потоков (при делении здесь необходимо округлять результаты «вверх»). Последние два параметра передают названия счетчиков стартовавших и завершившихся потоков соответственно.

Реализация для вычислительного кластера. Для выполнения клеточного автомата в вычислительном кластере на всех машинах, входящих в него, должна быть запущена среда из пакета CAME&L. Пусть на одной из машин (назовем ее главной) создается эксперимент (главный эксперимент), в процессе которого будут выполняться кластерные вычисления.

При выполнении инициализации эксперимента на главной машине всем остальным (или не всем, в зависимости от стратегии разделения задачи) посылается команда: создать вспомогательный эксперимент, который представляет собой часть главного, подзадачу. Причем состояние фрагмента решетки для данной подзадачи отправляется с учетом «полей», зон, которые не будут обрабатываться, но состояния клеток из которых могут потребоваться для вычисления новых значений для клеток основной области.

Каждый шаг на главной машине состоит в отправлении команд всем участникам эксперимента обновить поля (главная машина сообщает каждой, кто владеет информацией о состоянии клеток их полей) и произвести итерацию (шаг во вспомогательных экспериментах).

Необходимо отметить, что каждый эксперимент идентифицируется по так называемому дескриптору, в который входят строковое имя эксперимента (для его изменения можно выбрать в меню среды «Modeling» | «Change Rules ID...» или воспользоваться «горячим» сочетанием клавиш <Shift>+<F12>), адрес главной машины эксперимента и его номер. Данный дескриптор должен быть уникальным, чтобы однозначно определять эксперимент. Неповторимость строкового имени легко обеспечить на каждой машине в отдельности. В результате обеспечивается уникальность пары имени и адреса главной машины. Номер используется при вычислениях в кластере. Для главного эксперимента он равен нулю. Все вспомогательные эксперименты получают уникальные номера последовательно, начиная с единицы.

[1] Переменная pNetwork->GetThis()->uCPUsCount хранит число процессоров на данной машине.

Когда главная машина сочтет нужным, она может запросить у остальных состояние их подрешеток (исключая поля). В результате состояние всей решетки будет «собрано» на одной машине. После этого можно продолжить эксперимент или прекратить его.

Реализация автомата «Жизнь» для вычислительного кластера, как и для многопроцессорной системы, не сильно отличается от решения, приведенного в подразд. «Простейшая реализация». Потребуется внести следующие изменения.

Для удобства введем еще один параметр, p_iGather, целое число, определяющее количество шагов, по истечении которого необходимо собрать состояние всей решетки на главной машине. В случае, если он равен нулю, сбор не осуществляется. В конструкторе класса CALifeRules необходимо будет вызвать конструктор этого параметра, а также, как и в подразд. «Реализация с использованием зональной оптимизации» (с. 32), присвоить значение true переменной bFinalizeIfChanged для обеспечения целостности данных.

Функция SubCompute останется неизменной, однако функция Initialize существенно поменяется. Кроме того, потребуется определить функции Compute и Finalize. Три новые функции будут иметь следующий вид:

```
bool  CALifeRules::Initialize()
{
      // Проверка того, работает ли сетевой интерфейс
      if (!pNetwork->IsWorking()) return false;
      // Разделение задачи между машинами
      CreateCluster(1,1,true,2);
      // Ожидание, пока все участники кластера
      // подтвердят создание экспериментов и готовность
      return WaitForClusterReady();
}

void  CALifeRules::Finalize()
{
      // Удаление экспериментов на всех участниках
      DestroyCluster();
}

bool  CALifeRules::Compute()
{
      // Осуществление итерации
      return ComputeCluster(p_iGather.Get() &&
            (Step+1-pCluster->Step>=p_iGather.Get()));
}
```

Как видно из фрагмента приведенного кода, кластерные вычисления организуются с помощью пакета CAME&L чрезвычайно просто ввиду наличия богатого набора средств, содержащихся в классе CARules.

Опишем две переменные и две функции, члены класса CARules, использованные выше.

• public Network* pNetwork – переменная хранит указатель на описание сетевого контекста и интерфейса рабочей станции.

• public Cluster* pCluster – переменная хранит указатель на менеджер кластера.

• public virtual unsigned int CreateCluster(unsigned int overlap, unsigned char type, bool usethis, int remoteness) – функция разделяет задачу на подзадачи, отбирает машины, участвующие в эксперименте, и рассылает команды для создания вспомогательных экспериментов. Функция имеет следующие параметры:

unsigned int overlap – определяет величину полей;

unsigned char type – определяет способ разделения задачи. В данный момент поддерживаются следующие значения:

- 0 – отладочный способ, делящий решетку на вертикальные полосы, количество которых определяется параметром remoteness. Все части эксперимента создаются на главной машине;
- 1 – разделение на равные вертикальные полосы. Их количество зависит от числа участвующих машин;
- 2 – разделение на вертикальные полосы, пропорциональные факторам производительности машин («throughput factor»)[1]. Их количество зависит от числа участвующих машин;
- другие значения могут быть приняты пользователем для обозначения разнообразных стратегий разделения при переопределении данной функции;

bool usethis – определяет, учитывать ли главную машину при разделении задачи. Если параметр равен true, то на ней тоже должен быть создан вспомогательный эксперимент;

int remoteness – определяет удаленность машин, которые следует привлекать к эксперименту (в настоящее время не поддерживается).

Функция может быть переопределена в потомке.

• public virtual bool ComputeCluster(bool gather=false) – функция посылает всем участникам эксперимента команды для осуществления итераций и ожидает ответов об их завершении. Параметр gather определяет, нужно ли собрать состояние всей решетки после данной итерации или нет. Функция может быть переопределена в потомке.

Во фрагменте кода, приводимом выше, при определении того, нужно ли собирать состояние решетки, использовалось выражение «Step+1-pCluster->Step>=p_iGather.Get()». Здесь значение переменной Step – номер шага в главном эксперименте. К нему прибавлена единица, так как внутри функции Compute его увеличение еще не произошло. Значение переменной pCluster->Step – номер шага, на котором осуществлялась последняя сборка состояния всей решетки. Если эти две величины расходятся не

[1] Величина фактора производительности определяется средой для каждой машины, на которой она запускается. Пользователь может самостоятельно определить его, выбрав в меню среды «Tools» | «Workstation Characteristics...».

меньше, чем на значение параметра p_iGather, то состояние всей решетки будет собрано.

Реализация для обобщенных координат. Компонент правил для клеточного автомата «Жизнь» в обобщенных координатах [4] даже проще, чем приведенный в подразд. «Простейшая реализация», так как задача становится одномерной. Функции Initialize и SubCompute примут следующий вид:

```
bool  CALifeRules::Initialize()
{
      if (!p_iAlive.IsAnalyzed()) return true;

      DATUM(CAGenBoolDatum);
     iq=0;

      for(CACell c=datum->z.a1; c<=datum->z.b1; c++)
    {
          if (datum->Get(c)) iq++;
    }
      p_iAlive.Set(iq);

      return true;
}

bool  CALifeRules::SubCompute(Zone& z)
{
      DATUM(CAGenBoolDatum);

      CACell neig[12]; // Массив для хранения всех соседей
      // Переменная, хранящая количество соседей
      unsigned short ncount=GET_METRICS->GetNeighboursCount();
      unsigned char alive; // Количество живых соседей текущей клетки
     iq=0;

      for(CACell c=z.a1; c<=z.b1; c++)
         {
            alive=0;

               // Получение всех соседей и размещение их в массиве
               pUnion->Metrics->GetNeighbours(c,neig);

               // Подсчет числа живых соседей текущей клетки
              for (int k=0; k<ncount; k++) {
                   if (datum->Get(neig[k])) alive++;
             }

               // Применение функции переходов и определение
               // значения анализируемого параметра
              if (datum->Get(c)) {
                     datum->Set(c,!p_bDie[alive]->Get());
                    if (!p_bDie[alive]->Get()) iq++;
             } else {
                     datum->Set(c,p_bBorn[alive]->Get());
                    if (p_bBorn[alive]->Get()) iq++;
             }
         }
       // Присваивание значения анализируемому параметру
       p_iAlive.Set(iq);

      return true;
}
```

В дополнение к этому необходимо изменить условия совместимости данного компонента, так как он требует обобщенной метрики. Таким образом, выражение его условий примет вид «Data.bool.*&Metrics.2D.generalized.*».

Для постановки эксперимента теперь придется использовать обобщенную метрику (например, стандартный компонент «Square Grids Generalized») и соответствующее хранилище данных (стандартный компонент «Booleans for Generalized»).

Необходимо отметить, что компонент, реализованный в подразд. «Простейшая реализация», тоже может работать с обобщенной метрикой, если указать условия совместимости «Data.bool.*&Metrics.2D.*». Этого не было сделано изначально только для ясности изложения.

Таким образом, описанный в настоящем подразделе компонент лишь оптимизирован под работу с одномерными координатами, однако принципиально новой функциональности он не реализует.

Решение уравнения теплопроводности

Теперь приведем пример решения физической задачи – уравнения теплопроводности с помощью пакета CAME&L. Как известно, уравнение имеет вид

$$\frac{\partial T}{\partial t} = a\Delta T, \qquad (1)$$

где $a = \frac{\lambda}{\rho c_p}$; λ – коэффициент теплопроводности материала; ρ – плотность материала; c_p – теплоемкость материала при постоянном давлении.

Каждая клетка автомата будет хранить значение температуры в соответствующей точке пространства, представляемое числом с плавающей точкой. Таким образом, автомат будет моделировать поле температур.

Для постановки эксперимента снова потребуется двумерная решетка из квадратов (хотя может использоваться и любая другая двумерная решетка), картезианская метрика. Помимо этого необходимо воспользоваться хранилищем вещественных данных для картезианских координат (стандартный компонент «Doubles for Cartesians»), а соответствующие правила требуется разработать.

Класс, реализующий эти правила, назовем CATCERules. Дабы не загромождать реализацию физическими величинами, будем считать параметр a из выражения (1), а также величины dx, dy и dt равными единице. В результате в качестве функции переходов будем использовать следующее выражение в конечных разностях:

$$T'(x,y) = \\ = \frac{T(x+1,y)+T(x-1,y)+T(x,y+1)+T(x,y-1)}{4}. \qquad (2)$$

В случае, если для хранилища данных будет включено хранение состояния на двух последовательных шагах, то можно считать, что в левой части выражения (2) стоит температура на следующем временном шаге, по сравнению с теми, что стоят в правой части. Если хранение на двух шагах не включено, то значения температуры в обеих частях вы-

ражения считаются для одного и того же временного шага, а тогда порядок обхода клеток при вычислении новых состояний приобретает значение, хотя и не принципиальное.

Предусмотрим в компоненте правил четыре анализируемых вещественных параметра: p_dAverageT – средняя температура на решетке, p_dMaxT – максимальная температура на решетке, p_dMinT – минимальная температура на решетке, p_dAverSelT – средняя температура среди выделенных пользователем в среде клеток.

Описание класса CATCERules будет иметь следующий вид:

```
class CATCERules:public CARules
{
public:
    // Конструктор и деструктор
    CATCERules();
    virtual ~CATCERules() {};

    // Определение имени, описания, иконки и условий
    // совместимости
    COMPONENT_NAME(TCE Solution (plain))
    COMPONENT_INFO(Thermal conductivity equation solution rules)
    COMPONENT_ICON(IDI_ICON)
    COMPONENT_REQUIRES(Data.double.*&Metrics.2D.cartesian.*)

    // Следующие функций класса CARules будут переопределены
    virtual bool Initialize();
    virtual bool SubCompute(Zone& z);

public:
    // Анализируемые параметры
    ParamDouble p_dAverageT;
    ParamDouble p_dMaxT;
    ParamDouble p_dMinT;
    ParamDouble p_dAverSelT;

public:
    // Карта анализируемых параметров
    APARAMETERS_COUNT(4)
    BEGIN_APARAMETERS
        PARAMETER(0,&p_dAverageT)
        PARAMETER(1,&p_dMaxT)
        PARAMETER(2,&p_dMinT)
        PARAMETER(3,&p_dAverSelT)
    END_APARAMETERS
};
```

Функции Initialize и SubCompute могут быть реализованы следующим образом:

```
bool CATCERules::Initialize()
{
    // Инициализацию следует производить, только если хотя бы один из
    // параметров анализируется
    if (!p_dAverageT.IsAnalyzed() && !p_dMaxT.IsAnalyzed()
        && !p_dMinT.IsAnalyzed() && !p_dAverSelT.IsAnalyzed())
        return true;

    DATUM(CATypedDatum<double>);
    CACell c; // Текущая клетка
    double val=0.0; // Значение из текущей клетки

    // Переменные для вычисления значений анализируемых параметров
    double avt=0.0,avselt=0.0;
    double maxt=datum->Get(GET_METRICS->ToCell(
        datum->z.a1,datum->z.a2,0));
    double mint=datum->Get(GET_METRICS->ToCell(
        datum->z.a1,datum->z.a2,0));

    for(CACell i=datum->z.a1; i<=datum->z.b1; i++)
        for(CACell j=datum->z.a2; j<=datum->z.b2; j++)
    {
            // Получение идентификатора текущей клетки
            // по декартовым координатам
            c=GET_METRICS->ToCell(i,j,0);
            val=datum->Get(c);

            avt+=val;
            if (val<mint) mint=val;
            if (val>maxt) maxt=val;
            if (p_dAverSelT.IsAnalyzed()) {
                if (find(GET_GRID->SelCells[1].begin(),
                    GET_GRID->SelCells.end(),c)!=
                    GET_GRID->SelCells.end()) avselt+=val;
        }
    }
    // Присваивание значений анализируемым параметрам
    p_dAverageT.Set(avt/(int)datum->z.GetVolume());
    p_dMaxT.Set(maxt);
    p_dMinT.Set(mint);
    if (GET_GRID->SelCells.size())
        p_dAverSelT.Set(avselt/GET_GRID->SelCells.size()); else
        p_dAverSelT.Set(0.0);

    return true;
}

bool CATCERules::SubCompute(Zone& z)
{
    DATUM(CATypedDatum<double>);
    CACell c; // Текущая клетка
    double val=0.0; // Значение из текущей клетки

    // Переменные для вычисления значений анализируемых параметров
    double avt=0.0,avselt=0.0;
    double maxt=datum->Get(GET_METRICS->ToCell(
        datum->z.a1,datum->z.a2,0));
    double mint=datum->Get(GET_METRICS->ToCell(
        datum->z.a1,datum->z.a2,0));

    for(CACell i=z.a1; i<=z.b1; i++)
        for(CACell j=z.a2; j<=z.b2; j++)
    {
            // Получение идентификатора текущей клетки
            // по декартовым координатам
            c=GET_METRICS->ToCell(i,j,0);

            // Вычисление нового состояния клетки по формуле (2)
            val= datum->Get(GET_METRICS->ToCell(i-1,j,0));
            val+=datum->Get(GET_METRICS->ToCell(i+1,j,0));
            val+=datum->Get(GET_METRICS->ToCell(i,j-1,0));
            val+=datum->Get(GET_METRICS->ToCell(i,j+1,0));
            val/=4.0;
            datum->Set(c,val);

            avt+=val;
            if (val<mint) mint=val;
            if (val>maxt) maxt=val;
            if (p_dAverSelT.IsAnalyzed()) {
                if (find(GET_GRID->SelCells.begin(),
                    GET_GRID->SelCells.end(),c)!=
                    GET_GRID->SelCells.end()) avselt+=val;
        }
    }
    // Присваивание значений анализируемым параметрам
```

[1] Переменная SelCells класса CAGrid хранит список клеток, выделенных пользователем. Таким образом, данное условие, накладываемое на результат, возвращаемый функцией find, позволяет определить, принадлежит ли клетка c ко множеству выделенных или нет.

```
    p_dAverageT.Set(avt/(int)z.GetVolume());
    p_dMaxT.Set(maxt);
    p_dMinT.Set(mint);
    if (GET_GRID->SelCells.size())
        p_dAverSelT.Set(avselt/GET_GRID->SelCells.size()); else
        p_dAverSelT.Set(0.0);

    return true;
}
```

Как видно из приведенного фрагмента кода, реализация функции `Initialize` отличается от `SubCompute` только отсутствием в ней вычисления нового значения клетки и наличием проверки, анализируется ли хотя бы один из параметров.

Здесь приведено простейшее решение уравнения теплопроводности. Однако, по аналогии с игрой «Жизнь», легко разработать варианты с зональной оптимизацией для многопроцессорной и кластерной вычислительных систем или обобщенных координат. Кроме того, нетрудно ввести параметры, определяющие физические характеристики системы, упомянутые в выражении (1).

Заключение

Примеры решения шести задач иллюстрирует тот факт, что библиотека CADLib предоставляет разработчику богатейший инструментарий. При работе с программным обеспечением CAME&L знание языка программирования C++ позволяет решать широкий спектр задач.

Последний тезис может показаться странным, так как владение языком C++ само по себе позволяет решать вычислительные задачи, не применяя специализированных пакетов. Кроме того, это – нетривиальный навык, рассчитывать на наличие которого у пользователя не правомерно.

Однако, во-первых, организация параллельных и распределенных вычислений с помощью любого из существующих средств требует от пользователя существенных программистских навыков. Во-вторых, пакет CAME&L берет на себя реализацию огромного количества вспомогательной функциональности – от сетевого взаимодействия до визуализации и сохранения экспериментов, что в полной мере оправдывает его использование.

В-третьих, программный интерфейс библиотеки CADLib построен из настолько общих соображений, что позволяет, например, разработать компонент правил, который в качестве параметра принимал бы описание клеточного автомата на неком специализированном языке, как FORTH [6, 7] или CARPET [8]. В результате, конечный пользователь будет избавлен от необходимости владеть языком C++.

Примеры, приведенные в разделе «Реализация игры "Жизнь"», наглядно показали, что функция `SubCompute` практически не меняется при переходе от одной вычислительной системы к другой. Таким образом, функциональность базового класса `CARules` может быть расширена так, что реализация правил для однопроцессорной, многопроцессорной и кластерной вычислительных систем будет осуществляться одним компонентом. Однако такой класс не входит в библиотеку CADLib, чтобы оставить пользователю большую свободу при реализации своих решений, схем разделения задачи и т. п.

Исходные коды всех обсуждаемых в настоящей работе компонентов доступны на сайте [9]. Там же выложен весь пакет CAME&L.

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. **Gardner M.** The Fantastic Combinations of John Conway's New Solitaire Game «Life» // Scientific American. 1970. № 223.
2. **Гарднер М.** Математические досуги. М.: Мир, 1972.
3. **Гарднер М.** Крестики-нолики. М.: Мир, 1988.
4. **Naumov L.** Generalized Coordinates for Cellular Automata Grids // Computational Science – ICCS 2003. Part 2. Springer-Verlag. 2003.
5. **Гордеев А., Молчанов А.** Системное программное обеспечение. СПб.: Питер, 2001.
6. **Тоффоли Т., Марголус Н.** Машины клеточных автоматов. М.: Мир, 1991.
7. **Броуди Л.** Начальный курс программирования на языке FORTH. М.: Финансы и статистика, 1990.
8. **Spezzano G., Talia D.** Designing Parallel Models of Soil Contamination by the CARPET Language. 1998.
9. Сайт «CAMEL Laboratory» – http://camellab.spb.ru.

УДК 621.856

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ТЕОРИИ ГИБКИХ ТЕХНОЛОГИЧЕСКИХ СИСТЕМ И ИХ ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

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

Работа направлена на решение фундаментальной проблемы создания глобальных моделей социумов для проверки управленческих решений и их последствий без экспериментирования на людях и природе. 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. Вторая часть статьи содержит основы теории гибких дискретных технологических систем, позволяющей построить формальную модель 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 second part of article contains the base of the theory of the flexible discrete technological systems, allowing to constructing formal model of Human–Culture–Technology–Nature system.

Описание целей системы и теоретико-множественной модели гибких дискретных технологических систем

Для формального описания гибких дискретных технологических систем (ГДТС) используем теоретико-множественный подход М. Месаровича и Я. Такахары [1], который базируется на теории множеств и реляционной алгебре [2]. Модель системы M может быть задана как совокупность множеств характеристик системы $M = (A, B, C, ...)$. В общем случае $M = \{A_i / I = 1...I\}$, где I – число характеристик, A_i – характеристические множества. Тогда все множество состояний системы S_M может быть представлено как декартово произведение всех этих множеств: $S = (\times A_i / i = 1...I)$. Любое состояние системы s_k может быть описано как выборка из декартова произведения этих множеств: $(a_1, a_2, a_3, ..., a_{I-1}, a_I)$.

Выбор характеристических множеств всегда целесообразно начинать с определения целей существования и жизнедеятельности описываемой системы. В первой части статьи была введена реляционная структура цели как кортеж:

$$C = \langle d, \varepsilon, \theta, \tau_н, \tau_к \rangle,$$

где d – тип целевого (потока); ε – количество объектов типа d; θ – способ выполнения изменений свойств исходного объекта до получения его целевого состояния; $(\tau_н, \tau_к)$ – период, за который необходимо достигнуть цели. Множество целей на период T задается таблицей целевых заданий:

$$C_T = \{\langle d_l, \varepsilon_l, \theta_l, \tau_{нl}, \tau_{кl} \rangle / \, l = 1, L_T\},$$

где L_T – число элементарных целей, заданных на период T. Если взять объединение всех возможных C_T, то получим множество $\boldsymbol{C}$ целевых заданий, на которые должна быть рассчитана ГДТС. Следовательно, $\boldsymbol{C}$ является подмножеством декартова произведения доменов таблицы

$$C \subset D \times E \times \Theta \times T_н \times T_к,$$

при этом D – домен, определяющий множество возможных типов целевых объектов (потоков); E – домен разнообразия количественных характеристик по различным типам целевых объектов; Θ – домен, определяющий множество техпроцессов, на которые должна быть рассчитана исполнительная подсистема (ИПС); $T_н$ – множество моментов возможных начал выполнения целевых заданий; $T_к$ – множество моментов необходимых окончаний целевых заданий.

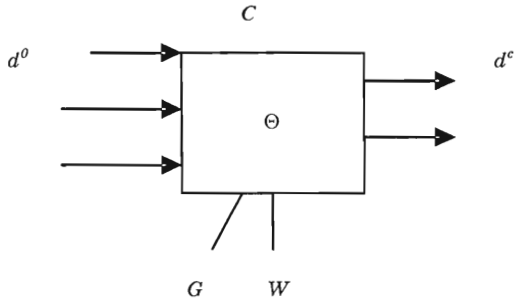
Таким образом, для описания ГДТС эти домены должны войти в число характеристических множеств ее теоретико-множественной модели:

$$M = (\boldsymbol{C}, D, \Theta, \boldsymbol{T}).$$

Технологический процесс Θ (способ изменения свойств $d^0{}_l$ для получения целевого объекта d_l) задается множеством операций: $\Theta_l = \{ O_{lj} / j = 1, J_l \}$, где O_{lj} – j-я операция l-го техпроцесса, обеспечивающая изменения каких-либо свойств d_l. Для выполнения каждой операции потребуются определенные исполнительные ресурсы r_{lj} и время t_{lj}. Множество $\{t_{lj}\}$ пополнит характеристическое множество $\boldsymbol{T}$, а домен ресурсов $\{t_{lj}\}$ из таблицы всех техпроцессов Θ составит характеристическое множество ресурсов системы R. Однако для различных систем не все выборки будут реализуемы, и чтобы определить допустимые состояния, необходимо задать множество ограничений Φ, при которых должны достигаться цели, и критериев F, по которым можно судить о качестве способов их достижения. Теперь можно написать полное теоретико-множественное представление статической модели ГДТС

$$M = (\boldsymbol{C}, D, \Theta, R, \Phi, F, \boldsymbol{T}).$$

Этот набор является инвариантом теоретико-множественного представления для систем класса ГДТС, на основе которого можно строить базы данных систем.



■ *Рис. 1. Структура компоненты ГДТС: цель C – объекты обработки (целевые потоки); d^0, d^c – технологические процессы Θ; O – характеристики операций; G – исполнительные средства; W – вспомогательные средства*

Описание формальной статической модели компоненты

Предметная область любой компоненты описывается универсальной структурой (рис. 1).

Характеристика элементарной цели уже приведена ранее:

$$C_l = \langle d_l, \varepsilon_l, \theta_l, \tau_{нl}, \tau_{кl} \rangle.$$

Характеристика объектов обработки d_l задается структурой

$$Xd_l = \langle d_l, \{FF\}_l \rangle,$$

где $\{FF\}_l$ – таблица с гибкой структурой.

Строка таблицы с гибкой структурой $FF = \langle IA, ZA, EI \rangle$ (таблицу с задаваемым именем атрибута в каждой строке назовем *гибким фреймом*); IA – имя атрибута (свойства) объекта, характеризующего d_l; ZA – значение атрибута, характеризующего d_l; EI – единицы измерения или код, определяющий метрику данного свойства, так как различные атрибуты будут иметь свои единицы измерения.

Технологический процесс (для получения целевого объекта d_l с заданными свойствами) задается множеством операций:

$$\Theta_l = (O_{lj} / j = 1, J_l),$$

где O_{lj} – j-я операция l-го техпроцесса изменения каких-либо атрибутов d_l из структуры Xd_l.

Характеристика операции задается структурой

$$XO_j = \langle O_j, G_m, W_p, U_{jmp}, t_{jmn} \rangle,$$

где G_m – группа однородных исполнительных средств, способных выполнить O_j; W_p – группа однородных вспомогательных средств, необходимых для исполнительных средств группы G_m при выполнении операции O_j; U_{jmp} – способ выполнения операции O_j, т. е. алгоритм изменения значений атрибутов из структуры Xd_l исполнительным средством из группы G_m с помощью вспомогательного средства из группы W_p; t_{jmn} – время, требуемое на выполнение операции O_j исполнительным средством из группы G_m с помощью вспомогательного средства из группы W_p.

Технологический процесс можно задать прямо в виде таблицы характеристик операций:

$$\Theta_l = \{\langle O_{lj}, G_m, W_p, U_{jmp}, t_{jmn} \rangle\} j = 1, J_l,$$

где J_l – число операций l-го техпроцесса.

Характеристика исполнительных средств задается реляционно-иерархической структурой

$$XG_m = \langle G_m, N_m, \{\langle g_{mn}, k_н, k_{пр}, k_{mn} \rangle\} \rangle,$$

где N_m – количество однородных исполнительных средств в группе G_m; g_{mn} – имя конкретного исполнительного средства из G_m; $k_н$ – коэффициент на-

дежности конкретного исполнительного средства g_{mn}; $k_{пр}$ – коэффициент производительности конкретного исполнительного средства g_{mn}; k_{mn} – координата конкретного исполнительного средства из G_m.

Характеристика вспомогательных средств задается реляционно-иерархической структурой

$$XW_p = \langle W_p, Q_p, \{\langle w_{pq}, \{FF\}\rangle\}\rangle,$$

где Q_p – количество вспомогательных средств в группе W_p; $\{FF\}$ – таблица, описывающая свойства вспомогательного средства w_{pq} из группы W_p с помощью гибкого фрейма FF. Использование гибкого фрейма вызвано многообразием имен атрибутов и их количеством при описании вспомогательных средств, необходимых основному средству для выполнения операции.

Таким образом, статическую модель любой компоненты или подсистемы ГДТС можно описать инвариантом из 6 реляционно-иерархических структур, которые легко сводятся к 9 реляционным, т. е. девятью типами таблиц. Все многообразие различий в свойствах и операциях техпроцессов (законов функционирования) разнообразных разнородных компонент, которые описываются профессионалами различных областей знаний в своих специфических моделях, заключено в атрибуте U_{jmp} *характеристики операции* O_j, *т. е. алгоритме изменения значений атрибутов ZA из структуры Xdl исполнительным средством из группы* G_m *с помощью вспомогательного средства из группы* W_p. При создании компьютерной имитационной модели эти алгоритмы разрабатываются независимо для каждой операции каждого техпроцесса каждой компоненты специалистами (технологами) предметной и проблемной области этой компоненты и системного аналитика-программиста и закладываются в динамические библиотеки, организованные по типам компонент и задач. Следовательно, разработка имитационной модели может вестись независимо параллельно большим коллективом различных специалистов-экспертов и аналитиков-программистов, что чрезвычайно важно при создании моделей систем класса Человек–Культура–Технологии–Природа (ЧКТП), таких как крупные фирмы, корпорации, производственные системы, территориальные образования, города, государства, экосистемы. При этом, только если все компоненты будут разрабатываться на едином принципе инварианта описания компонент, все они легко поймут друг друга, потому что не важно, каким образом изменялись атрибуты потоков, которыми обмениваются подсистемы и компоненты, важно, чтобы все понимали, в каких атрибутах каких структур в какой форме надо читать требующуюся информацию. Любая компонента становится прозрачной по структуре информации для всех других компонент любой подсистемы на любом уровне иерархии.

Формальное описание динамической модели компоненты и системы

Принимая решения в социумах или экосистемах, трудно предвидеть последствия принимаемых решений в процессе их реализации при функционировании системы. Имитационные модели предназначены для экспериментального исследования процесса функционирования систем в тех случаях, когда экспериментировать на системе невозможно или очень опасно. Для представления динамических процессов, протекающих в моделируемых системах, необходимо создание инвариантного описания динамической модели компонент и их взаимодействия в процессе функционирования всей системы, чтобы иметь возможность при создании имитационной модели объединять единым способом разрабатываемые независимыми экспертами компоненты и подсистемы.

Динамические параметры предметной области могут быть описаны следующими структурами:

таблицами сигналов об изменении состояний (о событии) и управляющих сигналов;

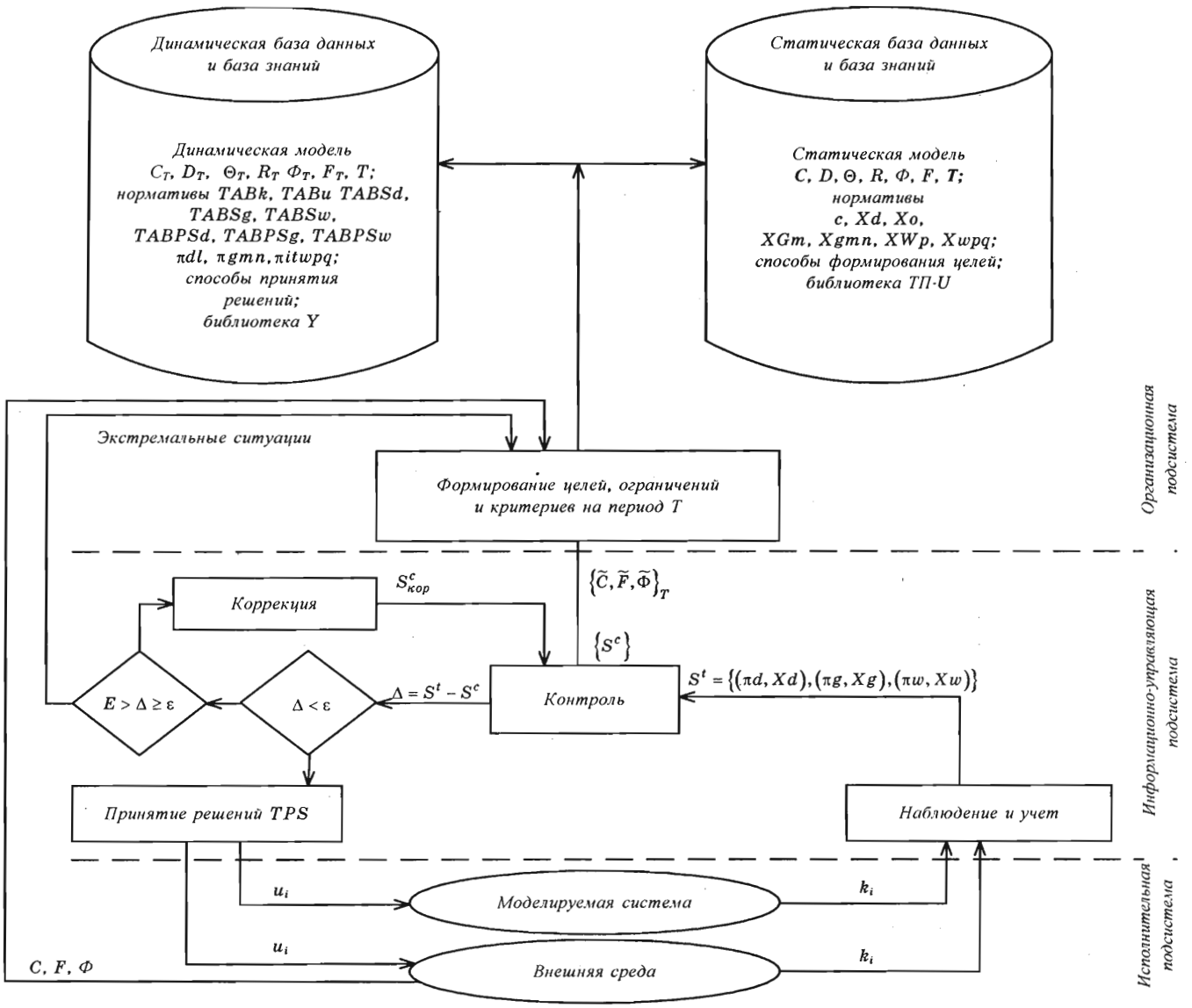
таблицами возможных состояний объектов обработки, исполнительных и вспомогательных средств;

■ ***Таблица Tabki***

№	Наименование	Код *ki*
1	Начат техпроцесс	*k1*
2	Завершен техпроцесс	*k2*
3	Начата подготовка	*k3*
4	Завершена подготовка	*k4*
5	Начата операция	*k5*
6	Завершена операция	*k6*
7	Начат ремонт (исправление)	*k7*
8	Завершен ремонт	*k8*
9	Сбой (отказ)	*k10*

■ ***Таблица Tabui***

№	Наименование	Код *ui*
1	Начать техпроцесс	*u1*
2	Завершить техпроцесс	*u2*
3	Начать подготовку	*u3*
4	Завершить подготовку	*u4*
5	Начать операцию	*u5*
6	Завершить операцию	*u6*
7	Начать исправление (ремонт)	*u7*
8	Завершить ремонт	*u8*



■ ***Рис. 2. Структура системы управления моделью и процессом моделирования***

таблицами переходов состояний (ТПС) объектов обработки, исполнительных и вспомогательных средств;

портретами объектов обработки, исполнительных средств, вспомогательных средств, описывающих их текущее мгновенное состояние.

Сигналы о событии в компонентах.

Для описания взаимодействия всех компонент, подсистем между собой и системы с внешней средой введем структуру сигналов о событиях. Событием в системе считается любое изменение в атрибутах d, g, w, связанное с выполнением технологического процесса или наличием сбоев и отказов. Сигнал, несущий сообщение о том, что у компоненты g_{mn} произошло событие типа k_i в момент τ, задается кортежем отношения $k = \langle g_{mn}, k_i, \tau \rangle$. Значения кодов сигналов k_i задаются таблицей *Tabki* с кортежем заголовка *Tabki* = ⟨наименование, код⟩. Аналогично с таким же кортежем задается таблица управляющих сигналов *Tabui*, вырабатываемых блоком принятия решений (рис. 2).

Таблица возможных состояний. Возможные состояния задаются относительно любой запланированной операции или техпроцесса (начата операция, идет операция, завершена операция, сбой и т. д.).

Для задания возможных состояний d относительно запланированной операции следует заполнить таблицу *TABSd*. Строка состояний *Sd* задается в виде логического слова из четырех логических параметров:

$$P1 = \begin{cases} 1, \text{ если техпроцесс начат;} \\ 0, \text{ если техпроцесс не начат;} \end{cases}$$

$$P2=\begin{cases}1, \text{ если операция выполняется;}\\ 0, \text{ если операция не выполняется;}\end{cases}$$

$$P3=\begin{cases}1, \text{ если операция завершилась браком;}\\ 0, \text{ если операция завершилась без отклонений;}\end{cases}$$

$$P4=\begin{cases}1, \text{ если техпроцесс завершен;}\\ 0, \text{ если техпроцесс не завершен.}\end{cases}$$

Строка таблицы возможных состояний для d задается кортежем

$TabSd = \langle наименование, Si, P1, P2, P3, P4\rangle$.

■ ***Таблица*** *TabSd*

i	Наименование	*Si*	*P1*	*P2*	*P3*	*P4*
0	Начальное	*S0*	0	0	0	0
1	Техпроцесс начат	*S1*	1	0	0	0
2	Операция начата (идет обработка)	*S2*	1	1	0	0
3	Операция завершена	*S3*	1	0	0	0
4	Операция завершена неудачно (брак)	*S4*	1	0	1	0
5	Техпроцесс завершен	*S5*	1	0	0	1
6	Техпроцесс завершен неудачно (с браком)	*S6*	1	0	1	1

В таблице приведено минимально необходимое число возможных состояний, однако оно зависит от решаемой задачи, и при реализации конкретной модели число строк таблицы может быть увеличено без необходимости изменения программы инвариантного ядра.

Для групп исполнительных средств G_m зададим возможные состояния логическими параметрами:

$$P1=\begin{cases}1, \text{ если исполнительное средство исправно;}\\ 0, \text{ если исполнительное средство неисправно;}\end{cases}$$

$$P2=\begin{cases}1, \text{ если исполнительное средство занято;}\\ 0, \text{ если исполнительное средство не занято;}\end{cases}$$

$$P3=\begin{cases}1, \text{ если исполнительное средство}\\ \text{готово к операции;}\\ 0, \text{ если исполнительное средство}\\ \text{не готово к операции;}\end{cases}$$

$$P4=\begin{cases}1, \text{ если исполнительное средство работает;}\\ 0, \text{ если исполнительное средство не работает.}\end{cases}$$

Таблица возможных состояний для G_m задается тем же кортежем, что и для *TabSd*.

■ ***Таблица*** *TabPSd*

i	Наименование состояния	*S*	*P1*	*P2*	*P3*	*P4*
0	Начальное	*S0*	0	0	0	0
1	Исправен	*S1*	1	0	0	0
2	Идет подготовка (наладка) к операции	*S2*	1	1	0	0
3	Подготовка завершена	*S3*	1	1	1	0
4	Идет обработка	*S4*	1	1	1	1
5	Обработка завершена	*S4* (*S3*)	1	1	1	0
6	Сбой (брак, отказ, нештатная ситуация)	*S5*	0	1	1	0
7	Идет ремонт (восстановление)	*S6*	0	1	0	0

Аналогично можно задать *TABSWp* для вспомогательных средств:

$TabSW = \langle наименование, Si, P1, P2, P3, P4\rangle$.

Законы переходов состояния. Зададим законы переходов из одного состояния в другое объектов обработки, групп исполнительных и вспомогательных средств (d, G_m, W_p) в виде таблиц переходов состояний

$$TABPS = \langle S_{old}, k_i, S_{new}, Y, u_i\rangle,$$

где S_{old} – старое состояние, в котором находился объект до наступления события, о котором сообщил сигнал k_i; S_{new} – новое возможное состояние, в которое должен перейти объект в результате события; Y – алгоритм (закон) перехода, отражающий в атрибутах и структурах изменения, произошедшие в системе в результате события; u_i – тип вырабатываемого управляющего сигнала о следующем назначаемом событии.

Например: $TABPSd = \langle S_{old}, k_i, S_{new}, Y, u_i\rangle$ для объектов обработки.

■ ***Таблица*** *TabPSd*

a	S_{old}	*ki*	S_{new}	*Y*	*ui*
0	*S0*	*k1*	*S1*	*Y0_1*	*u3*
1	*S1*	*k3*	*S2*	*Y1_3*	
2	*S2*	*k6*	*S3*	*Y2_4*	*u3*
3	*S3*	*k3*	*S2*	*Y3_3*	
4	*S3*	*k2*	*S5*	*Y3_2*	
5	*S2*	*k10*	*S4*	*Y2_10*	
6	*S4*	*k10*	*S6*	*Y5_10*	*u7*

Аналогичные таблицы переходов состояний должны быть заданы для G_m и W_p в соответствии с их таблицами возможных состояний *TabSGm* и *TabSWp* соответственно.

Портреты объектов обработки, исполнительных средств, вспомогательных средств.

■ *Таблица TabPSGm*

S_{old}	ki	S_{new}	Y	ui
S0	k0	S1	Y0_1	u3
S1	k3	S2	Y1_3	
S2	k4	S3	Y2_4	u5
S3	k5	S4	Y3_5	
S4	k6	S3	Y4_6	u3 (u5)
S1	k10	S5	Y1_10	u7
S2	k10	S5	Y2_10	u7
S3	k10	S5	Y3_10	u7
S4	k10	S5	Y4_10	u7
S5	k7	S6	Y5_7	u3
S6	k8	S1	Y6_8	u3 (u5)

Портреты предназначены для обеспечения возможности в любой момент времени в процессе функционирования системы наблюдать текущее состояние каждой компоненты системы.

Портрет объекта обработки πd задается кортежем

$$\pi d = \langle d_l, O_j, g_{mn}, w_{pq}, \varepsilon, \varepsilon^*, \varepsilon^{бр}, U_{mnj}, \tau_{нl}, \tau_{кl}, \tau_{нl}^{ф}, \tau_{кl}^{ф}, S\rangle,$$

где ε – число объектов обработки, заданное целевым заданием C_l; ε^* – число обработанных объектов типа d_l к текущему моменту; $\varepsilon^{бр}$ – число неудачно завершенных обработок объекта d_l(брак); U_{mnj} – способ выполнения операции (алгоритм изменения значений атрибутов из XO_j); $\tau_{нl}$, $\tau_{кl}$ – планируемые моменты начала и окончания операции O_j; $\tau_{нl}^{ф}$, $\tau_{кl}^{ф}$ – фактические моменты начала и завершения операции O_j; S – текущее состояние из списка возможных состояний объектов обработки относительно операции.

Портреты исполнительного πg и вспомогательного πw средств характеризуются соответственно кортежами:

$$\pi g = \langle g_{mn}, O_j, d_l, w_{pq}, \tau_{нl}, \tau_{кl}, \tau_{нl}^{ф}, \tau_{кl}^{ф}, S\rangle \text{ и}$$

$$\pi w = \langle w_{pq}, O_j, g_{mn}, d_l, Res, Rest, \tau_{нl}, \tau_{кl}, \tau_{нl}^{ф}, \tau_{кl}^{ф}, S\rangle,$$

где *Res*, *Rest* – полный и остаточный ресурс вспомогательного средства, если средства расходуемые.

Сигналы о событиях в моделируемой системе или внешней среде k_i поступают на блок формирования текущего состояния объектов моделирования (блок наблюдения, см. рис. 2), который производит изменения атрибутов портретов всех объектов, являющихся участниками события. Блок наблюдения обеспечивает возможность ЛПР иметь полную информацию в любой текущий момент о любом компоненте системы. Наблюдаемость системы определяется полнотой таблицы возможных сигналов относительно таблиц возможных (допустимых) состояний S всех типов объектов системы (d, g, w). Необходимость различимости состояний определяется целями, ограничениями и критериями решаемой задачи, т. е. зависит от компетентности экспертов, определяющих содержание всех таблиц.

Введенные структуры и два вида функций (U и Y) позволяют полностью задать предметную и проблемную область задачи имитационного моделирования.

Функция U_{jmp} из структуры характеристики операций $XO_{ij}=\langle O_{ij}, G_m, W_p, U_{jmp}, t_{jmp}\rangle$ отражает правила выполнения операции, т. е. способ изменения значений атрибутов в характеристике целевых объектов (потоков). Чтобы задать техпроцесс, надо сначала эксперту-технологу описать U_{jmp} для всех операций.

Правила обработки сигналов k_i по таблицам переходов состояний (Y) задают:

последовательность отображения изменений параметров состояния в портретах участников события (πd, πg, πw);

запуск процедуры U_{jmp} при сигнале окончания операции O_{ij} (фиксация изменения атрибутов объектов, исполнительных и вспомогательных средств в их гибком фрейме – *TABFF*);

поиск следующей операции и установка ее имени в портрет объекта обработки;

поиск свободных ресурсов в группах G_m и W_p, указанных в характеристике следующей операции, фиксация планируемой операции и ее исполнителей в портретах участников следующей операции;

формирование управляющего сигнала $u=\langle g_{mn}, u_i, \tau\rangle$ о необходимости исполнительному средству g_{mn} начать новую операцию.

Алгоритм обработки сигналов $k=\langle g_{mn}, k_i, \tau\rangle$ состоит из стандартной последовательности алгоритмов:

$$Y = Y_{inv} \oplus Y_{\tau} \oplus U_{jmp} \oplus F(u_i).$$

Y_{inv} – полностью инвариантная часть алгоритма, не зависящая ни от типа пришедшего сигнала k_i, ни от его источника g_{mn}, осуществляет поиск портретов всех участников операции, относительно которой произошло событие, и по ТПС фиксирует новое состояние в портретах πg_{mn}, πd_l, πw_{pq}.

Y_{τ} – инвариантная часть алгоритма, фиксирующая момент происшедшего события из кода сигнала $k.\tau$ в портреты участников πg_{mn}, πd_l, πw_{pq}.

U_{jmp} – алгоритм фиксации *результатов выполнения операции из структуры XO в структуре Xd*, если тип сигнала о событии $k.k_i$ является «завершением события».

$F(u_i)$ – алгоритм выработки управляющего сигнала u_i, выполняющий поиск следующей по техпроцессу операции для объекта предыдущей опе-

рации по результатам контроля правильности выполнения завершенной операции и наличию требуемых свободных ресурсов для нее.

Первые три алгоритма относятся к функциям блока наблюдения и учета, а четвертая выполняет функцию контроля соответствия текущего состояния целевому и принятия решений (см. рис. 2).

Запись алгоритмов в языке теории множеств и реляционной алгебры позволяет строго формализовать их до уровня общих операторов любых алгоритмических языков. В терминах операторов стандартных алгоритмических языков квантор общности $\forall$ (α) можно рассматривать как оператор цикла с заданным списком значений ($\forall\, \alpha(1...A)$ – «выполнять для всех значений списка»). Квантор существования может быть интерпретирован как условный оператор: $\exists\,(l')$:(P) – существует ли такое l', для которого выполняется предикат $P(l')$ или $\exists\,(P)(O)$ – если существует предикат P, то выполнить оператор O. Например, сформулируем несколько упрощенно алгоритм Y_{inv}.

Пусть пришел сигнал о событии $k = \langle g_{12}, k_2, 12\rangle$. Найдем портрет источника сигнала $k.g_{12}$, для чего необходимо в наборе портретов исполнительных средств $\pi g[m, n]$ найти тот, у которого значение поля $\pi_g.g_{mn}$ совпадает с полем $k.g_{12}$:

$$\forall\, m\,(m=1...M)\,\forall\;\; n\,(n=1...N)$$
$$(\exists\,(m', n')\,(\pi g.g_{mn}[m,n] = k.g_{12}).$$

В языках программирования это соответствует вложенным циклам по m и n, а во внутреннем цикле условный оператор проверяет выполнение условия $(\pi g.g[m,n] = k.g_{12})$. При его выполнении фиксируются текущие значения m и n:$(m' := m;\ n' := m;)$.

Теперь найденные координаты (m', n') строки таблиц портретов позволят получить портрет адресата $\pi g[m', n']$ и из него найти портреты второго и третьего участника событий:

$$\forall\, l\,(l = 1...L)\,(\exists\;\,(l')\colon (\pi d.d_l[l'] = \pi g.d_l[m', n']);$$

$$\forall\, p(p=1...P)\,\forall\, q(q=1...Q)(\exists\,(p', q')\colon(\pi w.w_{pq}[p', q'] = \pi g.w_{pq}[m', n']).$$

В трех найденных портретах $\pi g[m', n']$, $\pi d[l']$ и $\pi w[p', q']$ необходимо заменить старые значения состояний на новые по таблицам переходов состояния, для чего в таблицах переходов состояния для d_l требуется найти строку, удовлетворяющую следующим требованиям:

$$(\pi d.S[l'] = TABPSd.S_{old}[\alpha] \text{ и}$$
$$k.k_i = TABPSd.k_i[\alpha],$$

где α – индекс строк таблицы переходов состояния, а β и γ – индексы строк ТПС для g_{mn} и w_{pq} соответственно.

$$\forall\, \alpha(1...A)\,(\exists\;\,(\alpha')\colon (\pi d.S[l'] = TABPSd.S_{old}[\alpha]\ \&$$
$$k.k_i = TABPSd.k_i[\alpha]);$$

$$\forall\, \beta(1...B)\,(\exists\,(\beta')\colon (\pi g.S[m', n'] = TABPSg.S_{old}[\beta]\ \&$$
$$k.k_i = TABPSg.k_i[\beta]);$$

$$\forall\, \gamma(1...\Gamma)\,(\exists\,(\gamma')\colon (\pi w.S[p', q'] = TABPSw.S_{old}[\gamma]\ \&$$
$$k.k_i = TABPSw.k_i[\gamma]).$$

Остается занести новые состояния из найденных строк ТПС в поля S портретов $\pi g[m', n']$; $\pi d[l']$ и $\pi w[p', q']$:

$$\pi d.S[l'] := TABPSd.S_{new}[\alpha'];$$

$$\pi g.S[m', n'] := TABPSg.S_{new}[\beta'];$$

$$\pi w.S[p', q'] := TABPSw.S_{newd}[\gamma'].$$

Следующий алгоритм Y_τ должен проверить принадлежность сигнала либо к сигналам типа начал (начало техпроцесса, операции, ремонта, подготовки и т. д.), либо к сигналам типа окончания и занести в портреты участников события момент прихода сигнала $k.\tau$. При задании таблиц сигналов они разбиваются на два подмножества: KU – множество начал ($k_i \in KU$ нечетные) и KE – множество окончаний ($k_i \in KE$ четные). Тогда алгоритм в той же интерпретации представляется в следующем виде:

$$\exists\;(k.k_i \in KE)(\pi d.\tau_{кl}^{\Phi}[l'] := k.\tau)\;\Lambda$$
$$\exists\;(k.k_i \in KU)(\pi d.\tau_{нl}^{\Phi}[l'] := k.\tau);$$

$$\exists\;(k.k_i \in KE)(\pi g.\tau_{кl}^{\Phi}[m', n'] := k.\tau)\;\Lambda$$
$$\exists\;(k.k_i \in KU)(\pi g.\tau_{нl}^{\Phi}[m', n'] := k.\tau);$$

$$\exists\;(k.k_i \in KE)(\pi w.\tau_{кl}^{\Phi}[p', q'] := k.\tau)\;\Lambda$$
$$\exists\;(k.k_i \in KU)(\pi dw.\tau_{нl}^{\Phi}[p', q'] := k.\tau).$$

Запись приведена только как пример возможности описывать алгоритм формально и точно до типов операторов алгоритмических языков (для простоты опущены проверки тупиков и неоднозначности выходов).

Следующая составляющая алгоритма принятия решения в ответ на сигнал о событии в системе должна зафиксировать результаты, выражающиеся в тех атрибутах, свойствах и параметрах, которые изменились в процессе выполнения завершенной операции. Все они хранятся в структурах характеристик Xd, Xg, Xw в форме таблицы гибкого фрейма $TABFF = \{\langle IA, ZA, EI\rangle\}$. Это и позволяет объединять разнородные компоненты, функционирование которых описывается различными способами: таблицами результатов экспериментальных наблюдений за процессом функционирования (графиками, диаграммами, гистограммами и т. д.); расчетами по известным формулам в момент завершения операции; решением дифференциальных или алгебраических уравнений, логических или реляционных отношений, уравнений нелинейной динамики, в частных производных и т. д. Алгоритм U_{jmp} может описывать любыми способами любые процессы выполнения операций. Для общей модели важно лишь, чтобы они были выбраны и обоснованы высокопрофессиональными экспертами в конкретной области

знаний и исходные данные и результаты были представлены в структурах единой описанной модели. Алгоритмы функционирования компонент и подсистем U_{jmp} составляют собственную библиотеку для каждой конкретной системы и задачи и не входят в инвариантное ядро модели. Это и есть та часть модели, которая позволяет настраивать инвариант модели ГДТС на любую систему этого класса. Остальная настройка осуществляется просто заполнением значениями структур данных (баз данных) описанной модели.

Следующая часть алгоритма Y обработки сигнала о событии $k - F(u_i)$ отображает процесс принятия решения по выработке управляющего сигнала для выбора следующей операции. Количество возможных способов выбора управляющего воздействия зависит от оставшихся невыполненных целей и техпроцессов, ограничений, критериев и текущих состояний всех компонент системы. Так как функция наблюдаемости и учета изменения состояний исполнительного средства g_{mn}, объекта обработки d_l и вспомогательного средства w_{pq} уже выполнена в алгоритмах $Y_{inv} \oplus Y_\tau \oplus U_{jmp}$, то для принятия решения необходимо выполнить функцию контроля возможного рассогласования между целевым и текущим состоянием системы. Текущее состояние системы отражается в наборе портретов $S^t = \{\pi g, \pi g, \pi w\}$ и множестве характеристик Xd, Xg, Xw.

Целевые состояния можно задать в различных формах, например в виде расписания функционирования подсистем или компонент. Важно, чтобы структуры описания целевого (S^c) и текущего (S^t) состояний имели одинаковую метрику для реализации функции их сравнения и контроля. При использовании расписания в качестве последовательности целевых состояний это условие выполняется, так как структура его соответствует структуре динамических портретов. Тогда рассогласование на выходе блока контроля Δ (см. рис. 2) будет иметь структуру расписания

$$\Delta = \langle \Delta d, \Delta o, \Delta g, \Delta w, \Delta\varepsilon, \Delta t \rangle,$$

где $\Delta d, \Delta o, \Delta g, \Delta w$ – результат символьного сравнения фактического и планового значения типа целевого объекта (потока), операции, исполнительного и вспомогательного средства; $\Delta\varepsilon = \pi d.\varepsilon - \pi d.\varepsilon^*$ показывает разность между плановым и фактическим количеством целевого объекта (потока), а $\Delta t = \pi d.\tau_{кl}^{ф} - \pi d.\tau_{кl}$ – рассогласование фактического и планового времени завершения операции. Остается выбрать следующую операцию из расписания для того целевого объекта, у которого успешно завершилась операция, найти портреты запланированных ресурсов, внести изменения в их портреты (в случае, если у них за это время не произошли незапланированные события) и сформировать сигнал $u5$ «начать следующую операцию». Однако создание алгоритмов составления расписаний, расчет локальных и глобальных резервов, коррекций расписаний при небольших сбоях – все это сложные комбинаторные задачи, но они тоже содержат инвариантные части решений и настраиваемые на конкретные системы и ситуации алгоритмы, определяемые экспертами в конкретных областях знаний. Важно понять, что число возможных решений для каждого типа рассогласований определяется ограничениями и критериями, которые организационная подсистема (ОПС) задает организационно-управляющей подсистеме (ИУПС). Она должна задавать правила принятия решений и допустимость ситуаций. В случае экстремальной ситуации принятие решений переходит к ОПС, потому что только она может отменить какие-либо цели или увеличить ресурсы системы.

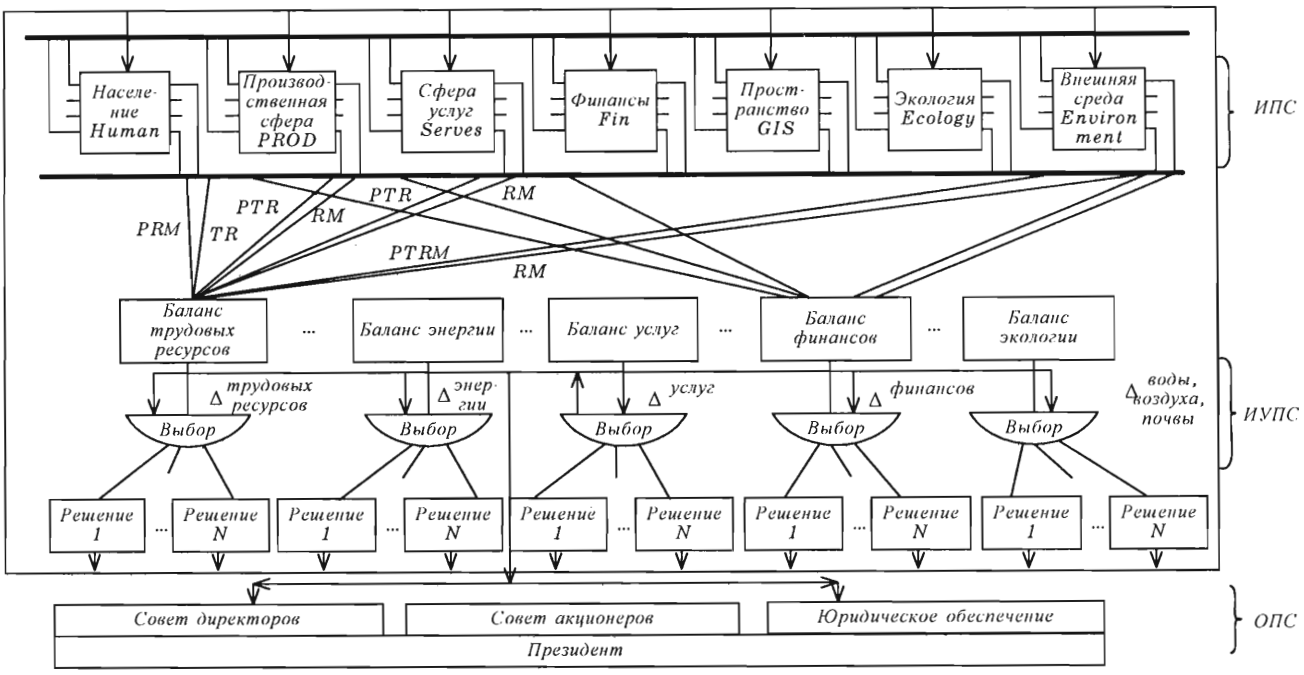
Объединение компонент в единую модель

Описание межкомпонентных связей в системе. Инвариантная структура описания исполнительной подсистемы ЧКТП содержит только самые общие подсистемы и потоки, которыми они обмениваются в процессе существования системы (рис. 3; подробнее см. рис. 5, ч. I). На самом верхнем уровне иерархии каждой подсистемы потоки могут обладать характеристиками, отражающими только самые общие их свойства. Например, подсистема «Население» (подсистема, обеспечивающая жизнедеятельность людей как социальных и биологических объектов любого социума) обладает потоком «Люди». Весь поток в целом можно характеризовать только количеством людей в потоке и общими показателями, характерными для всех людей или для всего потока. На этом уровне может решаться только задача замкнутости системы по наличию потоков из каждой подсистемы и потребностей в них других подсистем. Так как потоки различны по своей природе и имеют собственные метрики, то для проверки замкнутости системы и балансов ресурсов в процессе обмена различными потоками между подсистемами необходимо иметь общий эквивалент для сравнения. Его функцию и выполняет финансовый поток. Характеристика потоков любого уровня имеет инвариантный вид для всех потоков:

$$Xd^c = \{\langle IA, ZA, EI \rangle\} = TABFF.$$

Однако строки таблицы у всех потоков на верхнем уровне будут одинаковы. Например, *TABFF* для потока «Люди» подсистемы «Население»:

IA	*ZA*	*EI*
Имя потока *ID*	«Люди»	Строка
Количество в потоке ε	5000000	Целое число
Цена единицы потока cost		Число
Стоимость потока price		Число
Минимально необходимое количество min	4500000	Целое число
Потребность всех подсистем в этом потоке $\varepsilon^{тр}$	5100000	Целое число



■ ***Рис. 3.*** *Структура имитационной модели ЧКТП системы*

Или *TABFF* для основного целевого потока фирмы «Товары» из подсистемы «Производственная сфера»:

IA	*ZA*	*EI*
Имя потока *ID*	«Товары»	Строка
Количество в потоке ε	1000000	Целое число
Цена единицы потока cost	100	Число
Стоимость потока price	100000000	Число
Минимально необходимое количество min	500000	Целое число
Потребность всех подсистем в этом потоке $\varepsilon^{тр}$	950000	Целое число

Атрибут «Минимально необходимое количество» определяет уровень выпуска целевого потока каждой подсистемы, который еще обеспечивает выживание всей системы.

Так как число строк и наименование атрибутов для всех потоков одинаковы, то выгоднее использовать для представления структуры описания свойств потока на верхнем уровне не гибкий фрейм *FF*, а жесткий, заданный кортежем $Xd^{c} = \langle ID, \varepsilon, \text{cost}, \text{price}, \text{min}, \varepsilon^{тр} \rangle$. Этих данных будет достаточно для решения задачи проверки условий жизнеспособности целостной системы в начальный момент (при создании системы) и в любой текущий момент при ее функционировании. Введение стоимостного эквивалента каждого потока позволяет решать (отображать) все экономические задачи как статического, так и динамического типа, так как изменение цены единицы потока и его стоимости зависит от способа функционирования (Θ) подсистемы-источника потока и стоимости всех входных потоков из других подсистем. Такая модель позволяет ЛПР в одной подсистеме использовать знания экспертов о законах функционирования различных подсистем, выраженных в моделях, языках и теориях, незнакомых этому ЛПР, но видимых ему в структурах, отражающих результаты их жизнедеятельности. На рис. 3 приведена схема структуры связей трех подсистем ЧКТП-систем. На верхнем уровне находится ИПС, состоящая из семи подсистем. Результатом их функционирования являются события, изменяющие значения атрибутов выходных потоков. Информация о событиях поступает на блоки наблюдения и контроля информационно-управляющей подсистемы. Так как сравниваться могут только потоки, одинаковые по метрикам, то и балансных блоков должно быть столько, сколько имеется сравнимых потоков. Например, из подсистемы «Население» может поступить информация об изменении числа работающих (массовые увольнения в городе, «демографический провал» и т. д.). На блок баланса по трудовым ресурсам приходит информация о числе рабочих мест (*RM*) и потребности в трудовых ресурсах (*PTR*) из подсистем «Производственная сфера» и «Сфера услуг». Измеряются три рассогласования: количество трудовых ресурсов и рабочих мест (*RM-TR*), количество трудовых ресурсов и потребность в трудовых ресурсах (*TR-PTR*), количество рабочих мест и потребность в рабочих местах (*RM-PRM*). Эти рассогласования и порождают задачу (цель) системе управления: выработать управляющее воздей-

■ *Таблица* $XPS_1^{вых}$

№ потока l	Имя целевого потока d_{il}	Имя подсистемы-приемника потока PS_j	Функция отчисления $Xdl.кол$, %	Примечания
1	Трудовые ресурсы	Производственная сфера	70	На текущий момент 70% используются производственной сферой
1	Трудовые ресурсы	Сфера услуг	25	На текущий момент 25% используются сферой услуг
1	Трудовые ресурсы	Внешняя среда	5	5% уволилось из системы
2	Потребность в трудовых ресурсах	Внешняя среда	6	На 0,06% отдел кадров объявил вакансии
3	Потребность в финансовом потоке	Подсистема финансов	100	Подошел срок выплаты зарплаты

■ *Таблица* $XPS_1^{вх}$

№ потока l	Имя целевого потока d_{il}	Имя подсистемы-приемника потока PS_j	Функция отчисления $Xdl.кол$, %	Примечания
1	Люди	Население	30	Переместились на рабочие места внутри системы
1	Люди	Внешняя среда	25	Пришли в поисках работы
2	Потребность в трудовых ресурсах	Производственная сфера	9	Требуется замена уволенным
2	Потребность в трудовых ресурсах	Сфера услуг	25	Набирается новый отдел

ствие для уменьшения или устранения рассогласования. Для решения этой задачи необходима дополнительная информация о свойствах (атрибутах) потоков, требующих балансировки. Это означает, что для решения поставленной задачи в характеристику потока Xd должны быть добавлены новые атрибуты. Появление новых атрибутов потребует дополнения модели новыми компонентами, определяющими техпроцессы обработки новых атрибутов.

В качестве примера рассмотрим эту схему как модель фирмы. ИПС состоит из семи подсистем. Каждая подсистема имеет свой основной целевой поток и множество потоков потребностей, адресованных другим подсистемам, необходимым для формирования своего основного целевого потока. Если при создании фирмы правила функционирования всех подсистем были заданы таким образом, чтобы все потребности были удовлетворены, то на балансных блоках все рассогласования Δ должны быть в пределах возможных допусков ε.

Первая задача, которая должна решаться на имитационной модели, – проверка всех входных и выходных потоков на замкнутость. Не должно быть «висящих потоков» каждой потребности на входе подсистем, должен существовать целевой поток из какой-либо подсистемы (включая «Внешнюю среду»), удовлетворяющий эту потребность. Так как выходные целевые потоки одной подсистемы могут быть востребованы несколькими подсистемами, то необходимо ввести еще одну структуру, характеризующую подсистемы и их потоки с точки зрения законов потребления выходов одной подсистемы несколькими другими. Зададим их в форме реляционно-иерархического отношения.

Выходные потоки i-й подсистемы $PS_i(\forall\, i(i=1,7))$:

$$XPS_i^{вых} = \left\{ \left\langle \; d_{il}^{вых}, \left\{ \left\langle \; PS_j^{пр}, f\,(Xdl.кол) \; \right\rangle \right\} \; \right\rangle_{j\in J} \right\}_{l=1,L_i},$$

где $XPS_i^{вых}$ – характеристика взаимодействия подсистемы $PS_i\,(i=1,\,7)$ со своими приемниками по всем своим целевым потокам; $d_{il}^{вых}$ – целевые потоки подсистемы PS_i; $PS_j^{пр}$ – подсистемы-приемники; $f(Xdl.кол)$ – функция отчисления количества потока d_{il} для подсистемы PS_j.

Например, таблица, характеризующая связи подсистемы «Население» с другими подсистемами по целевым потокам «Трудовые ресурсы» (TR), «Потребность в финансовом потоке» (PF), «Потребность в трудовых ресурсах» (TR), отражена в таблице $XPS_1^{вых}$.

Такая же структура задается для всех входных потоков каждой подсистемы, так как в общем случае входные потоки могут складываться из выходных потоков нескольких подсистем:

$$XPS_i^{\text{вх}} = \left\{ \langle\, d_{il}^{\text{вх}}, \left\{ \langle\, PS_j^{\text{ист}}, f(Xdl.\text{кол}) \,\rangle \right\} \,\rangle_{j \in J} \right\}_{l=1,L_i},$$

где $XPS_i^{\text{вх}}$ – характеристика взаимодействия подсистемы PS_i со своими поставщиками; $d_{il}^{\text{вх}}$ – входные потоки для PS_i, характеристики которых описаны в структурах потоков поставщиков $PS_j^{\text{ист}}$.

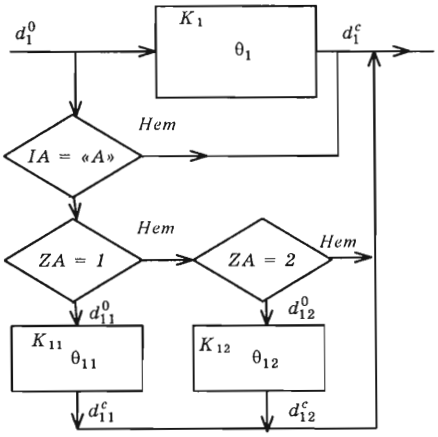
Например, из таблицы $XPS_1^{\text{вх}}$ видно, что на входе подсистемы «Население» поток «Люди» складывается из двух источников: «Внешней среды» и самой подсистемы «Население». Переход потока «Люди» с выхода подсистемы на собственный вход означает, что произошло событие, которое в результате изменило какие-то атрибуты потока (люди болеют, повышают квалификацию, перемещаются на новые рабочие места и в новом качестве снова входят в систему, изменяя свои правила функционирования).

Естественно, пример не соответствует конкретной системе. Информационные потоки на входе должны быть получены от множества различных экспертов из разных подсистем. Эксперты отделов кадров зададут общие требования ко всем сотрудникам данной фирмы. Эксперты-технологи зададут требование к трудовому ресурсу, способному обеспечить *XO.U* – алгоритм изменения свойств целевого объекта, потому что при внедрении новых технологий требуются специалисты принципиально других знаний и квалификации (что знает технолог, но плохо понимает ЛПР, судя по тому, что говорят о развитии высоких технологий при все ухудшающемся институте подготовки высококвалифицированных кадров по этим технологиям, а дело это небыстрое!). Эксперты-социологи, психологи, демографы и т. д. должны сформулировать личностные и социальные требования. Если компонента (трудовой ресурс) срабатывает не на цель системы, а на собственную цель («порадеть родному человечку», «откат», работа на более щедрого конкурента, назначение тепловика на место руководителя атомной станции и т. п.), то в систему закладываются потенциальные «Чернобыли», пожары, отключения энергосетей, катастрофы и т. д. Проиграть стратегию выбора кадров на имитационных моделях менее опасная стратегия формирования ИС для систем типа ЧКТП, особенно при создании ОПС, потому что можно увидеть последствия, к которым может привести наличие или отсутствие у претендента того или иного качества.

Итак, еще две структуры позволяют проверить взаимодействие подсистем по потокам между собой и системы с внешней средой по горизонтальным связям.

Общий принцип декомпозиции подсистем и компонент

Остается решить вопрос о едином способе декомпозиции целей, структур и функций в любой



■ ***Рис. 4.*** *Принцип декомпозиции компоненты*

подсистеме. Вспомним, что на верхнем уровне иерархии (на уровне подсистем) имеются только общие атрибуты целевых потоков, и, следовательно, для решения любой конкретной задачи нет информации о необходимых атрибутах. Адекватность модели определяется целью моделирования, ограничениями на способ моделирования и получения результата и критериями оценки результатов, в описании которых и задаются те атрибуты потоков, исполнительных и вспомогательных средств, которые необходимы для решения поставленной задачи. Как только определяются необходимые атрибуты, можно заполнить *FF*-фреймы всех потоков, исполнительных и вспомогательных средств и определить способы их обработки, задав характеристики операций *XOj*. Таким образом, получается новая компонента, которая в качестве своего целевого потока *d* имеет атрибут, подлежащий обработке, в качестве его техпроцесса θ – заданный алгоритм изменения его свойств, а в качестве исполнительных средств – средства, необходимые для изменения или получения новых значений *FF*-фрейма в *Xd.ZA*.

Декомпозиция компоненты показана на рис. 4. Поток $d_1{}^0$ обрабатывается компонентой K_1 с техпроцессом θ_1, при этом обнаруживается, что у части элементов потока $d_1{}^0$, обладающих атрибутом *IA* = «*A*», требуется дополнительная обработка в зависимости от значения атрибута *ZA*. Тогда надо проверить наличие имени атрибута «*A*» в *FF*-фрейме элементов потока $Xd_l{}^0$ и, если оно имеется, то проверять возможные значения этого атрибута. При этом поток будет разделяться на число подпотоков, соответствующее числу возможных различных значений атрибутов плюс 1, если возможны элементы потоков, не обладающих атрибутом с именем «*A*». Например, если *ZA* может принять только значения 1 или 2, то придется добавить новые две компоненты K_{11} и K_{12} с различными техпроцессами θ_{11}, θ_{12} обработки возникших подпотоков $d_{11}{}^0$ и $d_{12}{}^0$. В результате выполнения тех-

процессов будут получены два различных целевых потока $d_{11}{}^c$ и $d_{12}{}^c$, которые вместе с оставшейся частью потока, не обладавшей атрибутом с именем «A», составят целевой поток $d_l{}^c$ на выходе компоненты K. Такое разделение потоков возможно на любом уровне иерархии. Оно всегда связано с определенной целью, которая выделяет часть потока верхнего уровня, требует введения дополнительного атрибута и техпроцесса и порождает компоненты следующего уровня иерархии. Этот подход позволяет получить формальный метод декомпозиции и встраивания новых компонент в иерархические структуры любых подсистем в ОПС, ИУПС и ИПС сложных систем типа ЧКТП.

Заключение

Таким образом, основы общей теории систем типа ГДТС для построения имитационных моделей базируются на следующих принципах:

модель системы рассматривается всегда как целостная, состоящая из исследуемой системы и внешней среды (см. рис. 2, ч. I);

функциональная структура системы представляется тремя подсистемами: организационной, информационно-управляющей и исполнительной (см. рис. 3, ч. I);

каждая компонента системы на любом уровне иерархии задается инвариантом из 5 атрибутов (C, D, Θ, G,W) (см. рис. 1);

для описания любой компоненты системы определена математическая статическая и динамическая модель, состоящая из инвариантной и настраиваемой на конкретный тип системы части;

введен принцип декомпозиции подсистем по цели и атрибутам потоков, позволяющий формальным способом строить иерархические структуры подсистем и включать новые компоненты (см. рис. 4);

на основе декомпозиции цели существования системы в цели поведения (жизнедеятельности) выделен инвариант структуры исполнительной подсистемы систем типа ЧКТП.

Разработанные математические основы общей теории систем типа ГДТС могут служить хорошей основой программного инструментального средства для создания имитационных моделей и процесса имитационного моделирования сложных систем.

Литература

1. **Месарович М., Такахара Я.** Общая теория систем: математические основы. М.: Мир, 1978.
2. **Кодд Е. Ф.** Реляционная модель данных для больших совместно используемых банков данных. СУБД N1, 1995.
3. **Перовская Е. И.** Основные принципы построения моделей сложных гибких дискретных технологических систем // Системный анализ в проектировании и управлении: Тр. VIII Междунар. науч.-практ. конф. СПб.: Изд-во СПбГТУ, 2004. С. 126–130.
4. **Перовская Е. И.** Создание глобальных моделей социумов для проверки управленческих решений и их последствий // Системный анализ в проектировании и управлении: Тр. VII Междунар. науч.-практ. конф. СПб.: Изд-во СПбГТУ, 2003. С.186–190.
5. **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. C.194–196.
6. **Перовская Е. И., Фетисов В. А.** Автоматизация гибких дискретных систем. Л.: Изд-во ЛГУ, 1986. 183 с.
7. **Перовская Е. И.** Имитационные модели для поддержки принятия решений в системах Человек–Культура–Технологии–Природа // Системный анализ в проектировании и управлении: Тр. V Междунар. науч.-практич. конф. СПб.: Изд-во СПбГТУ, 2001. С. 177–181.

УДК 007: 57+007:573

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И БИОМЕХАНИЧЕСКИЙ АНАЛИЗ КРИТИЧЕСКОГО СОСТОЯНИЯ И КОРРЕКЦИИ СТРУКТУР СОСУДИСТОЙ СИСТЕМЫ (Часть I)

П. И. Бегун,
доктор техн. наук, профессор
О. В. Кривохижина,
аспирантка
Санкт-Петербургский государственный электротехнический университет
В. К. Сухов,
доктор мед. наук, заведующий отделением эндоваскулярной хирургии
Городская многопрофильная больница № 2, Санкт-Петербург

Разработаны компьютерные модели для исследования перемещений и напряжений при дилатации кровеносных сосудов с бляшками разной степени развития и определения критического состояния истинных мешотчатых аневризм. Проведенные исследования влияния геометрических параметров и механических свойств бляшки и сосуда на величину дилатируемого отверстия и выявление критического состояния аневризм предопределяют необходимость предоперационного анализа с использованием этих моделей.

Computer models and scheme of calculation for research of displacement and strains in true aneurysms are developed and at a dilatation of blood vessels with plaques of a different degree of development. Outcomes of the carried out evaluations confirm an imperative need of expansion of number of parameters at clinical diagnostics of a critical condition of aneurysms and preoperative diagnostics of outcomes endovascular surgical operations on stenosed blood vessels.

Введение

Главными причинами нарушения кровотока в сосудистой системе являются прогрессирующий атеросклероз и образование аневризм.

Важнейшие вопросы современной сосудистой хирургии – внедрение новых медицинских технологий, выбор технологии хирургического вмешательства в каждом конкретном случае и прогнозирование результатов этого вмешательства. Современный уровень развития науки и вычислительной техники позволяет решать задачи использования компьютерных программ для поддержки интеллектуальной деятельности врача. Такие программы дают возможность более эффективно выполнять диагностический процесс, прогнозировать тяжесть течения заболеваний, улучшать качество принимаемых врачом решений. Однако в лечебно-диагностическом процессе, как правило, не используются компьютерные программы ни для сбора и анализа медицинской информации, ни для поддержки деятельности врача как при выборе стратегии и тактики лечения больного, так и при разработке технологии проведения операций.

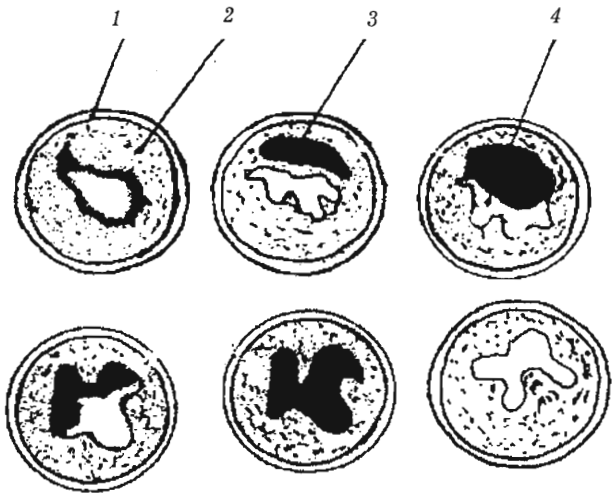
Наглядной иллюстрацией необходимости информационного обеспечения лечебно-диагностического процесса является отсутствие у врачей ряда специальностей убежденности в рациональности использования методов рентгенохирургии для выработки малотравматичных способов лечения, сочетающих в себе различные комбинации консервативного и инвазивного селективного воздействия на патологический очаг [1]. Существует масса противоречий в оценке целесообразности использования рентгенохирургических методов в лечении многих видов патологических процессов и отдельных заболеваний, относящихся как к кардиологии, так и к областям общей хирургии, нейрохирургии, урологии, гинеко-

логии и онкологии. В то же время рентгенохирургия, получившая развитие как прикладная специальность в сердечно-сосудистой хирургии, может решать многие задачи как самостоятельно, так и в содружестве с другими медицинскими направлениями. Определить эти взаимодействия на основе общих методологических принципов рентгенохирургических вмешательств в самых различных органах и системах возможно, только объединив усилия специалистов разных областей медицинских и технических знаний. Во всех принципиальных направлениях возникающих при этом проблем: медицинских, технических и фундаментальных – неотъемлемой частью является моделирование биологических объектов.

Математическое моделирование необходимо для понимания закономерностей функционирования биологических объектов в норме, патологии и при хирургических операциях и совершенствования методов ранней диагностики нарушений биомеханических свойств биологического объекта. Одним из основных условий математического моделирования биологического объекта является возможно более полный анализ и формализация той функции, которую он выполняет в норме и патологии в процессе жизнедеятельности организма или его соответствующей части.

В данной работе на примере моделирования и исследования напряженно-деформированного состояния (НДС) при дилатации кровеносных сосудов и при развитии аневризм проиллюстрирована возможность предоперационной диагностики эндоваскулярных хирургических операций по поводу устранения этих патологий.

В статье приняты следующие обозначения: σ – напряжение; N – число конечных элементов; L_c – длина сегмента сосуда; $L_{бл}$ – длина бляшки; L_a – длина аневризмы; R_c – радиус сосуда; $R_{бл}$ – радиус стенозированного отверстия; h_c – толщина стенки сосуда; h_a – толщина стенки аневризмы; H_a – высота аневризмы; ν – коэффициент Пуассона; $\boldsymbol{u}$ – перемещение; p – давление; E_c, $E_{бл}$, E_a – модуль нормальной упругости сосуда, бляшки, аневризмы соответственно.



■ *Рис. 1. Атеросклеротические бляшки поздней стадии развития:*
1 – стенка сосуда; 2 – бляшка; 3 – тромб; 4 – просвет артерии

Атеросклеротические поражения сосудистой системы и способы их устранения

Атеросклероз – хроническая болезнь, характеризующаяся липоидной инфильтрацией (накоплением жироподобных веществ) внутренней оболочки артерий эластического и смешанного типа с последующим развитием на стенке соединительной ткани. Основной морфологический элемент атеросклероза бляшка – очаговое утолщение внутренней оболочки артерии, возникающее в результате разрастания соединительной ткани в зоне отложения липидов. Атеросклероз часто поражает артерии, в структуре которых эластиновые волокна занимают видное место, определяя характер функционирования. Первичное повреждение эластиновой ткани приводит к освобождению и накоплению мукополисахаридов, которые легко соединяются с водой, липидами и белками. Это повышает гидратацию, приводит к набуханию артериальной стенки, делает ее рыхлой. Атеросклероз — результат хронического травмирования артериальной стенки ударами систологической волны крови из-за того, что защитная система артерий не в состоянии выполнить свое назначение. Подобное наблюдается в следующих случаях:

1) жесткие и твердые ткани оказываются в аномальной близости к артерии;

2) две артерии расположены очень близко друг к другу;

3) участок артерии направлен перпендикулярно к ответвлениям или изгибам артерий;

4) артерия растягивается до предела в условиях повышенного артериального давления, и внутренняя оболочка лежит под растянутой средней оболочкой, как на жестком основании [2, 3].

Вначале на месте атеросклеротической бляшки возникает сужение просвета артерии (стеноз), затем, при прогрессировании процесса, наступает полная окклюзия сосуда. На различных стадиях развития бляшки изменяются ее морфологические, геометрические и механические характеристики. Атеросклеротическая бляшка на ранней стадии развития имеет фибромускулярное строение. Она включает в свою толщу пролиферированные гладкомышечные элементы, коллаген, эластиновые волокна, а также интраклеточные и экстраклеточные липидные отложения. Атеросклеротические бляшки поздней стадии развития (рис. 1) отличаются от ранних фибромускулярных бляшек наличием некротических фокусов, кровоизлияний в толще бляшки, иногда очагами кальцификации. При этом обычно часть

внутренней эластичной мембраны повреждена (нарушена ее целостность) и затронут также медиальный слой стенки сосуда (рис. 2). Фиброзная крышка бляшки или организованный пристеночный тромб на ней сужают просвет артерии.

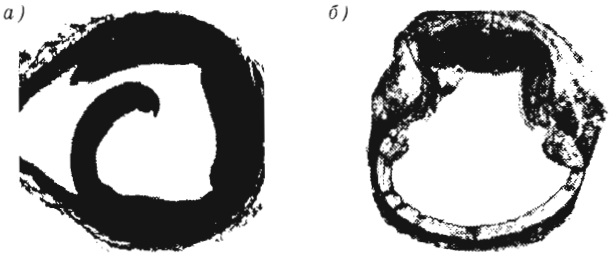
Для разрушения бляшек разработаны технологии эндоваскулярного воздействия: лазерная, ультразвуковая, роторная и ротоблатерная ангиопластики, ротационная атероэктомия и баллонная дилатация со стентированием [2]. Значительный прогресс в разработке технологии баллонной ангиопластики с последующим стентированием привел к существенному сокращению использования других технологий в клинической практике (с 18% от общего числа малоинвазивных эндоваскулярных рентгенохирургических операций на кровеносных сосудах в 1983 г. до 3% в 1999 г.). Около 97% операций проводят, используя технологию баллонной дилатации [2], когда воздействие на бляшку происходит при расширении баллона, установленного в зоне ее локального расположения [4].

При баллонной дилатации бляшек ранней стадии развития наблюдается растяжение сосудистой стенки и вдавливание массы бляшки в стенку артерии. После такого расширения стенки сосуда происходит деструкция гладкомышечных волокон медии и замещение их макрофагами. По мере заживления на месте механической травмы артерии контуры стенок артерии становятся вновь ровными или иногда как бы аневризматически расширяются [2]. Атеросклеротические бляшки поздней стадии развития являются самыми частыми объектами коронарной ангиопластики.

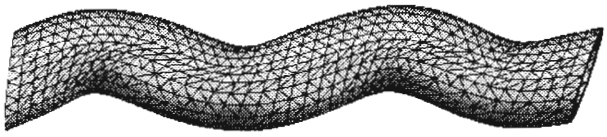
Для выбора необходимого давления при баллонной дилатации необходимо выполнить компьютерное моделирование и исследовать НДС в дилатируемом кровеносном сосуде с учетом особенностей расположения бляшек и формы дилатируемого сегмента сосуда.

Компьютерное моделирование и исследование напряжений и перемещений в дилатируемых сосудах

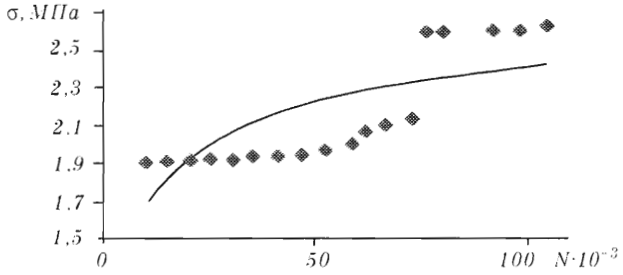
Расчетные схемы для исследования напряженно-деформированного состояния при дилатации кровеносных сосудов с осесимметричными и неосесимметричными бляшками, расположенными на прямолинейных и криволинейных сегментах сосудов, были построены при следующих допущениях: структура каждой стенки сосуда: интима, медия, адвентиция и бляшка – содержит по несколько разнонаправленных слоев с различной ориентацией волокон в них. Поэтому, при выраженных анизотропных свойствах структур стенки сосуда и бляшки, можно рассматривать материал всей стенки сосуда и материал бляшки как состоящий из неориентированных волокон и считать, что каждый из них в целом однородный и изотропный. Введено допущение о сплошности среды и отсутствии начальных напряжений в структурах сосуда.



■ ***Рис. 2.*** *Поврежденная интима кровеносных сосудов:*
а – отслоение и заворот интимы;
б – расколы в интиме

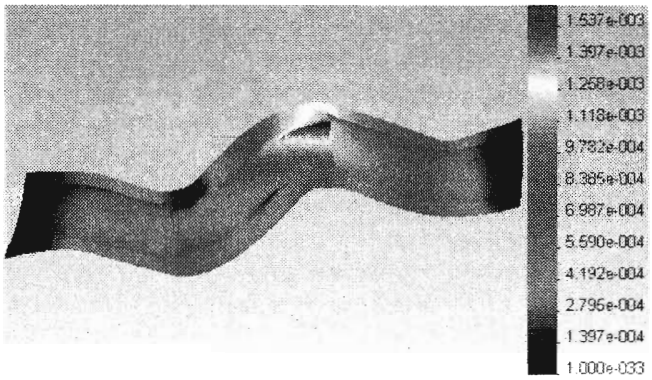


■ ***Рис. 3.*** *Конечно-элементная модель дилатации кровеносных сосудов*



■ ***Рис. 4.*** *Зависимость экстремальных значений напряжений σ в структурах дилатированного кровеносного сосуда от числа конечных элементов*

Математические модели построены в рамках механики трехмерного тела с помощью численного метода – метода конечных элементов (КЭ). Вычисления проведены при использовании параметрической программы Solid Works, предназначенной для создания геометрических моделей твердых тел, и программы Cosmos – для вычисления напряжения и деформаций в этих телах методом КЭ. Для анализа использованы тетраэдальные элементы. В параболических тетраэдальных элементах ребра криволинейные. При существенной кривизне ребер точность вычислений уменьшается. Линейные тетраэдальные элементы обеспечивают линейное изменение составляющих перемещений в пределах КЭ и определение их в направлении исходной системы координат. Поэтому при вычислениях приняты линейные тетраэдальные элементы (рис. 3). При этом учтено, что чем больше разница в размерах ребер КЭ, тем ниже точность вычислений и больше вероятность локальной расходимости решений. Погрешность расчета зависит от плотности сетки – размеров КЭ. Рис. 4 иллю-



■ ***Рис. 5.*** *Распределение перемещений* **u** *при дилатации криволинейного сегмента стенозированной внутренней сонной артерии с осесимметричным расположением бляшки:*
$L_c = 60$ мм, $L_{б1} = 30$ мм, $R_c = 3{,}5$ мм, $h_c = 1{,}5$ мм, $R_{б1} = 1{,}5$ мм, $E_c = 0{,}85$ МПа, $E_{б1} = 0{,}425$ МПа, $p = 10^6$ Па, $\nu = 0{,}4$

стрирует влияние числа КЭ на результаты расчета экстремальных значений напряжений при дилатации стенозированного кровеносного сосуда. Вычисления напряженно-деформированного состояния при дилатации сосудов проведены при разбиении структур на 80 тысяч тэтраэдальных конечных элементов (см. рис. 3).

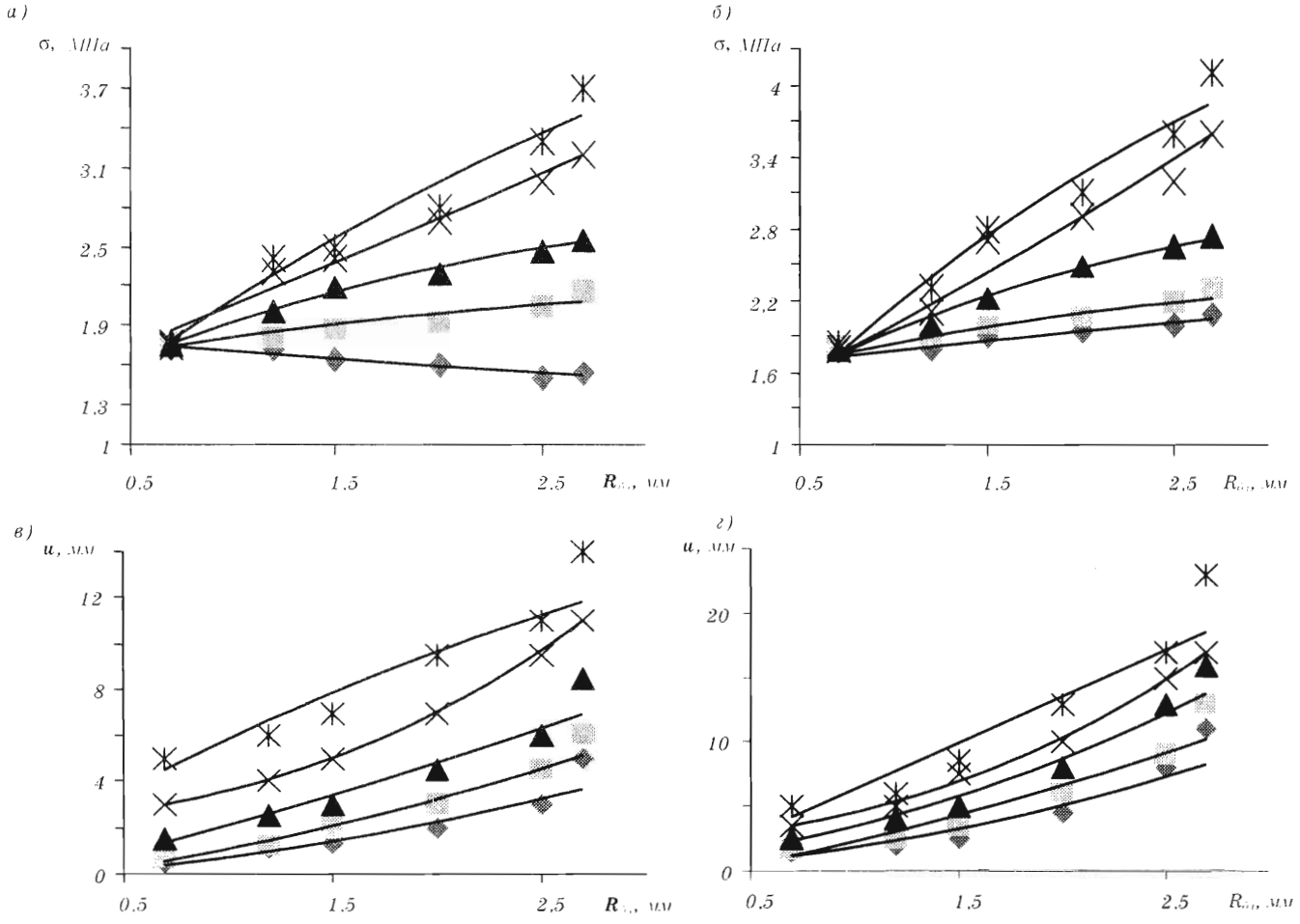
Сопоставим результат проведенной реконструкции стенозированной венечной артерии при баллонной дилатации с результатом вычислений по разработанным компьютерным моделям:

1) наружный и внутренний диаметры в норме соответственно 5 и 3 мм;

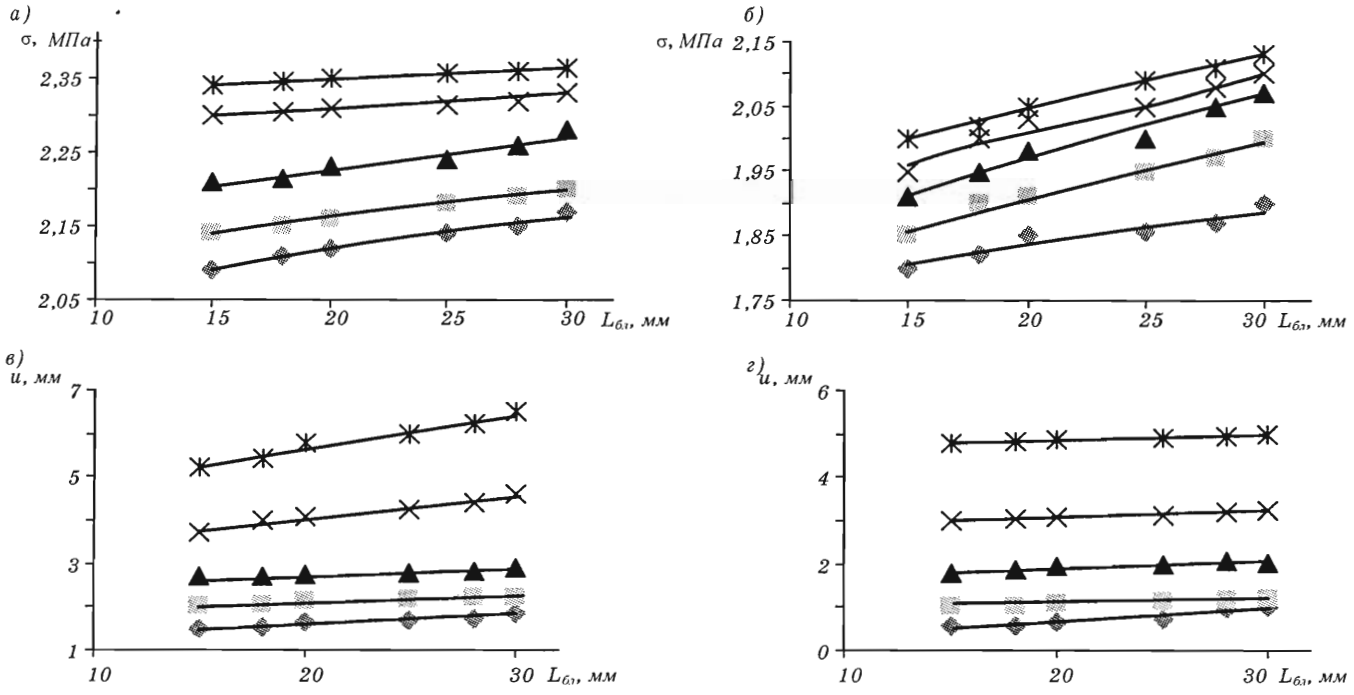
2) ось отверстия стенозированного сосуда смещена относительно оси сосуда в норме на расстояние 0,38 мм;

3) минимальный условный внутренний диаметр бляшки 0,75 мм, протяженность бляшки 15 мм;

4) модуль упругости сосуда и бляшки 1,45 и 0,7 МПа соответственно. В результате рентгено-



■ ***Рис. 6.*** *Зависимости экстремальных напряжений σ (а, б) и перемещений* **u** *(в, г) в прямолинейном (а, в) и криволинейном (б, г) сегменте внутренней сонной артерии с осесимметричным расположением бляшки от радиуса бляшки $R_{б1}$:*
$E_c = 0{,}85$ МПа; $L_c = 60$ мм; $L_{б1} = 30$ мм; $R_c = 3{,}5$ мм; $h_c = 1{,}5$ мм; $p = 10^6$ Па; $\nu = 0{,}4$
($E_{б1}$ (а, б): – 2,55; – 1,7; – 0,85; – 0,425; – 0,28 МПа;
$E_{б1}$ (в, г): – 0,28; – 0,425; – 0,85; – 1,7; – 2,55 МПа)



■ ***Рис. 7.*** *Зависимости экстремальных напряжений σ (а, б) и перемещений **u** (в, г) в прямолинейном (а, в) и криволинейном (б, г) сегменте внутренней сонной артерии с неосесимметричным расположением бляшки от длины бляшки $L_{бл}$:*
E_c = 0,85 МПа; L_c = 60 мм; R_c = 3,5 мм; e = 1 мм; $R_{бл}$ = 0,87 мм; h_c = 1,5 мм; p = 10^6 Па; ν = 0,4 ($E_{бл}$ (а, б): – 2,55; – 1,7; – 0,85; – 0,425; – 0,28 МПа; $E_{бл}$ (в, г): – 0,28; – 0,425; – 0,85; – 1,7; – 2,55 МПа)

хирургической операции при давлении в жестком баллоне 1,6 МПа внутренний диаметр дилатированного сегмента сосуда – 3 мм, по данным проведенных вычислений – 2,55 мм.

Заметим, что в биологических структурах необходимо учитывать не только допускаемые напряжения материала структуры, но и предельные напряжения, при которых структура сохраняет свои функции. Так, анализ баллонной дилатации венечных артерий показывает, что при напряжении, превышающем 0,61 МПа, межоболочечные пружинные конструкции медиального слоя раз-

Показатель	Осесимметричная бляшка		Неосесимметричная бляшка	
	ПС	КС	ПС	КС
$L_{бл}$ = 30 мм $R_{бл}$ = 0,87–3 мм	**Напряжение увеличивается**			
	1,05; 1,6	1,05; 1,8	1,01; 2	1,25; 2,5
	Перемещение увеличивается			
	7; 3,5	13; 6,6	3; 5	2,28; 9
$L_{бл}$ = 5–30 мм $R_{бл}$ = 0,87 мм	**Напряжение увеличивается**			
	1,05; 1,1	1,05; 1,2	1,02; 0,97	1,05; 1,08
	Перемещение			
	не изменяется	уменьшается 1,01; 1,3	не изменяется	уменьшается 1,01; 1,3

Примечания. 1. Приняты параметры: длина сосуда L_c = 60 мм, диаметр сосуда R_c = 3,5 мм, толщина стенки сосуда h_c = 1,5 мм, модули упругости: сосуда E_c = 0,85 МПа, бляшки $E_{бл}$ = 0,85–2,25 МПа; коэффициент Пуассона ν = 0,4.
2. ПС – прямолинейный сегмент, КС – криволинейный сегмент.
3. Напряжение и перемещение изменяется в разы.

рушаются, и артерии, сохраняя свою целостность, функционально не соответствуют норме [3].

Эпюры напряжений и перемещений представлены на рис. 5. Исследованы зависимости экстремальных напряжений и перемещений в структурах стенозированных сосудов от их геометрических параметров и механических свойств при дилатации сегментов с осесимметричными бляшками (рис. 6) и с неосесимметричными бляшками (рис. 7). На напряженно-деформированное состояние сосуда существенное влияние оказывает кривизна дилатированного сегмента (таблица).

Литература

1. **Сухов В. К., Шлойдо Е. А., Качанов И. Н.** К вопросу о современных подходах в лечении ИБС // Современные направления в диагностике, лечении и профилактике заболеваний. СПб.: Ольга, 2001. 184 с.
2. **Jean Marco, Gincarlo Biamino, Jean Fajadet, Marie Claude Morice.** The Paris course on revascularization// Europa organization. Paris, 2000. 441 p.
3. **Бегун П. И.** Гибкие элементы медицинских систем. СПб.: Политехника, 2002. 300 с.
4. **Бегун П. И, Сухов В. К.** Проблемы информационного обеспечения малоинвазивных интервенционных рентгено-хирургических операций на кровеносных сосудах // Информационно-управляющие системы. 2002. № 1. С. 52–56.
5. **Седов В. М., Богомолов М. С., Бабков А. А.** Аневризмы брюшного отдела аорты: Учеб. пособие/ СПбГМУ. СПб., 2001. 58 с.
6. Дооперационная оценка степени риска хирургического лечения больных с аневризмами брюшной аорты/ А. В. Покровский, В. Н. Дан, Ю. П. Богатов, А. М. Златовчен //Грудная и сердечно-сосудистая хирургия. 2003. № 1. С. 48–51.
7. Новое и старое во внутрисосудистом лечении аневризм сосудов головного мозга/ В. С. Папувцев, А. Ю. Иванов, А. В. Скупченко, Д. Е. Мацко // Регионарное кровообращение и микроциркуляции. 2003. Т. 2. С. 28–36.
8. Хирургические технологии в лечении аневризм грудного и торакоабдоминального отделов аорты/ Ю. В. Белов, А. Б. Степаненко, А. П. Генс и др. // Хирургия. 2003. № 2. С. 22–27.
9. **Белов Ю. В., Хамитов Ф. Ф.** Диагностика аневризм торакоабдоминального отдела аорты // Грудная и сердечно-сосудистая хирургия. 2001. № 3. С. 72–77.
10. Метод обходного временного шунтирования в хирургии аневризм грудного и торакоабдоминального отделов аорты/ В. С. Аракелян, С. П. Новиков, Н. Р. Гамзеев и др. // Хирургия сердца и сосудов. 2003. № 3. С. 47–52.
11. **Белов Ю. В., Чарчан Э. Р.** Клапаносохраняющие операции у больных аневризмой восходящего отдела аорты с аортальной недостаточностью// Грудная и сердечно-сосудистая хирургия. 2004. № 1. С. 59–64.
12. **Ледерле Ф., Сэмьюэль В., Джонсон Т.** Какой подход лучше при небольшой аневризме брюшного отдела аорты: безотлагательное иссечение или динамическое наблюдение?// Международный медицинский журнал. 2004. № 5. С. 497–503.
13. **Nevitt MP, Ballard DJ, Hallet JW Jr.** Prognosis of abdominal aortic aneurysms:a population-based study/ / N Engl J Med. 1989. № 321. P. 1009–1014.
14. **Glimaker H, Holmberg L, Elvin A** et al. Natural history of patients with abdominal aortic aneurysm// Eur J Vase Surg. 1991. № 5. P. 125–130.
15. **Бегун П. И., Шукейло Ю. А.** Биомеханика: Учебник для вузов. СПб.: Политехника, 2000. 463 с.

СВЕДЕНИЯ ОБ АВТОРАХ

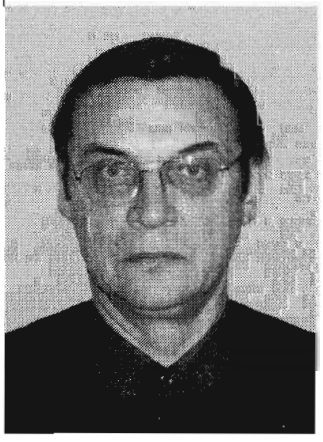
БЕГУН
Петр
Иосифович

Профессор, заместитель заведующего кафедрой прикладной механики и инженерной графики Санкт-Петербургского государственного электротехнического университета, академик академии медико-технических наук России.
В 1962 году окончил Ленинградский военно-механический институт.
В 2003 году защитил диссертацию на соискание ученой степени доктора технических наук.
Является автором 250 научных публикаций.
Область научных интересов – биомеханика человека.

КРИВОХИЖИНА
Оксана
Владимировна

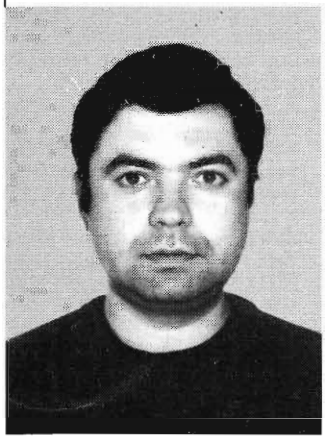
Аспирантка Санкт-Петербургского государственного электротехнического университета.
В 2004 году окончила Санкт-Петербургский государственный электротехнический университет.
Является автором 20 научных публикаций.
Область научных интересов – биология и механика.

НОВИКОВ
Федор
Александрович



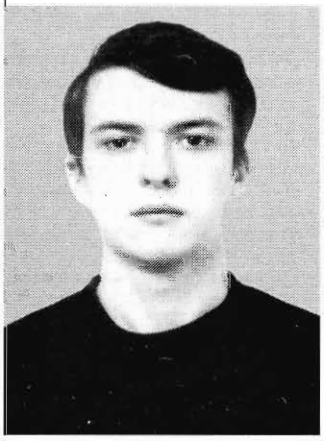
Доцент кафедры прикладной математики Санкт-Петербургского государственного политехнического университета.
В 1974 году окончил математико-механический факультет Ленинградского государственного университета.
В 1983 году защитил диссертацию на соискание ученой степени кандидата физико-математических наук.
Является автором 40 научных публикаций.
Область научных интересов – прикладная математика, технологии программирования.

ДАВИДЧУК
Андрей
Геннадьевич



Ассистент кафедры моделирования вычислительных и электронных систем Санкт-Петербургского государственного университета аэрокосмического приборостроения.
В 1998 году окончил Санкт-Петербургский государственный технический университет.
Является автором шести научных публикаций.
Область научных интересов – разработка математических моделей, технологии программирования, объектно-ориентированное проектирование.

НАУМОВ
Лев
Александрович

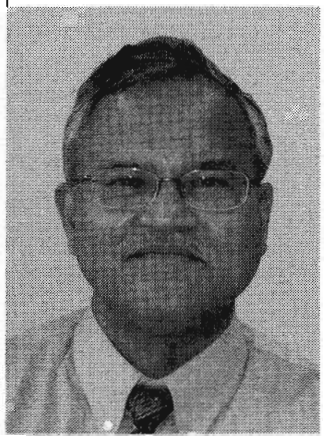


Аспирант Санкт-Петербургского государственного университета информационных технологий, механики и оптики.
Стипендиат Президента РФ.
Является автором 32 научных публикаций.
Область научных интересов – параллельные вычисления, клеточные автоматы, Grid – системы, вычислительные кластеры.

ПЕРОВСКАЯ
Евгения
Ивановна

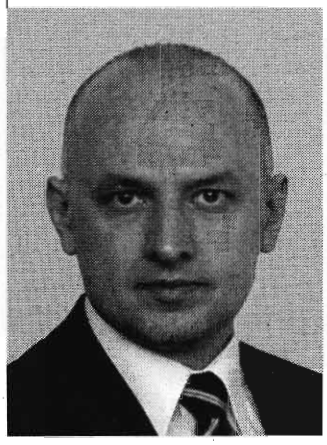
Профессор кафедры вычислительных систем и сетей Санкт-Петербургского государственного университета аэрокосмического приборостроения. Лауреат Премии Совета министров СССР за 1982 год.
В 1956 году окончила Ленинградский политехнический институт.
В 1982 году защитила диссертацию на соискание ученой степени 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».

СИНГХ Нишид



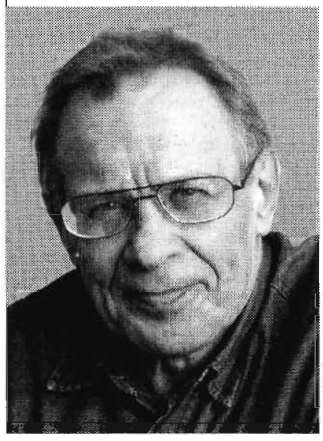
Начальник отдела руководства проектами фирмы ETRANS AG – координатора швейцарской передающей сети.
В 1976 году окончил Технический университет г. Канпур, Индия (IIT Kanpur India).
В 1989 защитил диссертацию на соискание ученой степени кандидата технических наук в Федеральном технологическом институте в Цюрихе (ETH Zurich).
Является автором 35 научных публикаций.
Область научных интересов – системы управления электрическими сетями и системы управления энергией (EMS).

ЧУБРАЕВ Дмитрий Валерьевич



Руководитель проекта разработки и внедрения системы оперативной безопасности швейцарской передающей электрической сети на фирме ETRANS AG, Швейцария.
В 1993 году с отличием окончил Санкт-Петербургский государственный технический университет.
В 1998 году защитил диссертацию на соискание ученой степени кандидата технических наук.
Является автором 12 научных публикаций.
Область научных интересов – информационные системы в электротехнике.

СУХОВ Валентин Константинович



Заведующий отделением эндоваскулярной хирургии городской многопрофильной больницы № 2 (Санкт-Петербург).
В 1965 году окончил Первый Ленинградский медицинский институт им. ак. И. П. Павлова.
В 1991 году защитил диссертацию на соискание ученой степени доктора медицинских наук.
Является автором более 200 научных публикаций.
Область научных интересов – кардиология, рентгенохирургия, кардиохирургия.

ШЕПЕТА Дмитрий Александрович

Доцент кафедры компьютерной математики и программирования Санкт-Петербургского государственного университета аэрокосмического при-боростроения.
Обладатель сертификата MCSD.
В 1997 году окончил Санкт-Петербургский государственный университет аэрокосмического приборостроения.
В 2000 году защитил диссертацию на соискание ученой степени кандидата технических наук.
Является автором 18 научных публикаций.
Область научных интересов – математическое моделирование и программирование.

УДК 621.391.826; 681.5

Математические модели эхо-сигналов кораблей, наблюдаемых локаторами бортовых систем обработки информации

Давидчук А. Г., Шепета Д. А. – Информационно-управляющие системы, 2005. – № 6. – С. 2–8.

Предлагается математическая модель эхо-сигналов кораблей, построенная на основе экспериментальных данных, которая позволяет учитывать флюктуации амплитуд и длительностей сигналов, а также их взаимные корреляционно-спектральные характеристики. Математическая модель основана на логарифмически нормальном многомерном законе распределения флуктуаций амплитуд и длительностей.

Список лит.: 14 назв.

UDK 621.391.826; 681.5

Mathematical models of ships' echo-signals detected by on board information processing systems

Davidchuk A. G., Shepeta D. A. – IUS, 2005. N 6. –P. 2–8.

The mathematical model of ships' echo – signals, constructed on the experimental data basis which allows to take into account fluctuations of amplitudes, durations of signals, and mutual correlation and spectral characteristics is offered. The mathematical model is based on the logarithmic-normal multidimensional law of distribution fluctuations of amplitudes and duration.

Refs: 14 titles.

УДК 602-507

Визуальное конструирование программ

Новиков Ф. А. – Информационно-управляющие системы, 2005. – № 6. – С. 9–22.

В статье описывается подход к проектированию и реализации набора компонентов, которые в совокупности составляют инструментальное средство, основанное на языке UML и поддерживающее все фазы процесса разработки приложений.

Список лит.: 22 назв.

UDK 602-507

Visual designing of programs

Novikov F. A. – IUS, 2005. N 6. –P. 9–22.

An approach to designing and implementing a set of components that form an instrumental tool, based on UML and supporting all phases of application development, is described in the article.

Refs: 22 titles.

УДК 621.31

Программный комплекс определения перегрузок на этапе краткосрочного планирования режима эксплуатации сети

Сингх Н., Чубраев Д. В. – Информационно-управляющие системы, 2005. – № 6. – С. 23–29.

Одним из элементов системы обеспечения оперативной безопасности электропередающих сетей Западной Европы, входящих в союз UCTE (Union for the Co-ordination of Transmission of Electricity – Союз по координации передачи электроэнергии, объединяющий 34 системных оператора Европы), является процесс DACF (Day-Ahead Congestion Forecast – Прогноз перегрузок на день вперед), отвечающий за предсказание перегрузок на этапе краткосрочного планирования. В статье описывается подход к автоматизации процесса DACF на фирме ETRANS AG, являющейся координатором швейцарской передающей сети и энергетического блока «Юг» Европейской объединенной сети.

Список лит.: 7 назв.

UDK 621.31

Software complex of overloads definition at a stage of short-term planning of network operation mode

Singh N., Chubraev D.V. – IUS, 2005. N 6. –P. 23–29.

One of the important elements of operational security system of the European interconnected power network is the DACF (Day-Ahead Congestion Forecast) process, that is to be performed by all countries-members of UCTE (Union for the Co-ordination of Transmission of Electricity). This article describes the approach of Swiss Transmission System Coordinator ETRANS AG to the automation of the DACF process.

Refs: 7 titles.

УДК 681.3.06:62-507

Решение задач с помощью клеточных автоматов посредством программного обеспечения CAME&L (Часть II)

Наумов Л.А. – Информационно-управляющие системы, 2005. – № 6. – С. 30–38.

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. Приводится пример решения физической задачи, уравнения теплопроводности.

Список лит.: 9 назв.

УДК 621.856

Математические основы теории гибких технологических систем и их имитационное моделирование

Перовская Е.И. – Информационно-управляющие системы, 2005. – № 6. – С. 39–50.

Работа направлена на решение фундаментальной проблемы создания глобальных моделей социумов для проверки управленческих решений и их последствий без экспериментирования на людях и природе. 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. Вторая часть статьи содержит основы теории гибких дискретных технологических систем, позволяющей построить формальную модель систем Человек–Культура–Технологии–Природа.

Список лит.: 7 назв.

UDK 681.3.06:62-507

Tasks solution with using bit-mapped automatic devices by means of software CAME&L (Part II)

Naumov L.A. – IUS, 2005. N 6. –P. 30–38.

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: 9 titles.

UDK 621.856

The system analysis and imitating modeling as means of classical and exact sciences results unification

Perovskaja E.I. – IUS, 2005. N 6. –P. 39–50.

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 second part of article contains the base of the theory of the flexible discrete technological systems, allowing to constructing formal model of Human–Culture–Technology–Nature system.

Refs: 7 titles.

УДК 007: 57+007:573

Компьютерное моделирование и биомеханический анализ критического состояния и коррекции структур сосудистой системы (Часть I)

Бегун П. И., Кривохижина О. В., Сухов В. К. – Информационно-управляющие системы, 2005. – № 6. – С. 51–56.

Разработаны компьютерные модели для исследования перемещений и напряжений при дилатации кровеносных сосудов с бляшками разной степени развития и определения критического состояния истинных мешотчатых аневризм. Проведенные исследования влияния геометрических параметров и механических свойств бляшки и сосуда на величину дилатируемого отверстия и выявление критического состояния аневризм предопределяют необходимость предоперационного анализа с использованием этих моделей.

Список лит.: 15 назв.

UDK 007: 57+007:573

Computer modelling and biomechanical analysis of the critical condishion and correction of the vascular system (Part I)

Begun P. I., Krivohizhina O.V., Suhov V. K. – IUS, 2005. N 6. –P. 51–56.

Computer models and scheme of calculation for research of displacement and strains in true aneurysms are developed and at a dilatation of blood vessels with plaques of a different degree of development. Outcomes of the carried out evaluations confirm an imperative need of expansion of number of parameters at clinical diagnostics of a critical condition of aneurysms and preoperative diagnostics of outcomes endovascular surgical operations on stenosed blood vessels.

Refs: 15 titles.

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.

СОДЕРЖАНИЕ ЖУРНАЛА «ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ» за 2005 г. [№ 1–6]

CONFERENCE
ММТТ 19

Министерство образования и науки Российской Федерации
Воронежская государственная технологическая академия
Воронежский государственный технологический университет
Российский химико-технологический университет
Московский государственный университет инженерной экологии
Московская государственная академия тонкой химической технологии
Ангарская технологическая академия и другие ведущие вузы и НИИ РФ

МЕЖДУНАРОДНАЯ НАУЧНАЯ КОНФЕРЕНЦИЯ "МАТЕМАТИЧЕСКИЕ МЕТОДЫ В ТЕХНИКЕ И ТЕХНОЛОГИЯХ - ММТТ-19"

Место проведения конференции: Воронежская государственная технологическая академия.
Адрес: 394000, Россия, Воронеж, проспект Революции, 19.

30 мая – 1 июня 2006 года

ЦЕЛИ КОНФЕРЕНЦИИ

Научная конференция ММТТ-19 проводится с целью подведения итогов применения математических методов в технике и технологиях и обсуждения современных направлений математического и компьютерного обеспечения технологических и технических систем.

НАПРАВЛЕНИЯ РАБОТЫ КОНФЕРЕНЦИИ

Качественные и численные методы исследования дифференциальных уравнений
Математические вопросы оптимизации и оптимального управления технологических процессов
Математическое моделирование технологических процессов
Математическое моделирование и проектирование экологически безопасных технологических процессов
Компьютерная поддержка производственных процессов (синтез, конструирование и проектирование, надежность и безопасность)
Интеллектуальные системы в технике, технологиях и медицине
Математические методы и задачи в экономических, гуманитарных науках и медицине
Математическое моделирование информационно-измерительных систем
Информационные технологии в образовании

В рамках конференции ММТТ-19 на базе Кисловодского гуманитарно-технического института **26 – 28 сентября 2006 г.** проводится **Региональная Южная конференция ММТТ-19** и **Международный научно-методический симпозиум «Современные проблемы многоуровневого образования»** в г. Ростов-на-Дону и г. Геленджик на базе Донского государственного технического университета (ДГТУ) 27–30 сентября 2006г.

Одновременно с конференцией ММТТ-19 состоится **Международная школа молодых ученых (ШМУ-11)**, на которой ведущими учеными будут прочитаны лекции и организованы секции:
Информатизация технических систем и процессов
Интеллектуализация управляемых систем и процессов
Автоматизация технических систем и процессов
Научные работы магистрантов и магистров

КОНТРОЛЬНЫЕ СРОКИ

Научные доклады и сообщения, представляемые на конференцию ММТТ-19, ШМУ-11 и Симпозиум должны содержать новые результаты. Каждый автор может представить на конференцию и Симпозиум не более 3 докладов, на ШМУ не более 4-х сообщений. Машинописные тексты докладов (2 – 6 с.) и сообщений (1 – 2 с.) представляются в одном экземпляре в Оргкомитет ММТТ до 15 февраля 2006 г. (105066, Москва, Ст. Басманная, 21/4, МГУИЭ, кафедра ТКА, Балакирев В. С.); электронные копии докладов направляются в два адреса по электронной почте в формате RTF по e-mail: mmtt19@ mail.ru; mmtt19@yandex.ru.

Дополнительная информация и справки

Москва: (095) 267-12-67, Балакирев Валентин Сергеевич;
(095) 434-71-11, Юловская Виктория Дмитриевна
Воронеж: (0732) 55-46-12, Хаустов Игорь Анатольевич,
haustov@vgta.vrn.ru

Информацию о конференции и Симпозиуме можно найти на сайте ВГТА: http://www.vgta.vrn.ru