ISSN 2222-8772

# Математические Структуры и Моделирование

Выпуск 23
2011

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Ф.М. ДОСТОЕВСКОГО
ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК

# МАТЕМАТИЧЕСКИЕ СТРУКТУРЫ И МОДЕЛИРОВАНИЕ

**Выпуск 23**

Омск 2011

Журнал «**Математические структуры и моделирование**» / Омск : Омский государственный университет, 2011. – Вып. 23. – 113 с.

ISSN 2222-8772 (print)
ISSN 2222-8799 (online)

---



---

**Адрес научной редакции**

---

Россия, 644053, Омск-53, ул. Грозненская, 11
Омский государственный университет
факультет компьютерных наук
E-mail: lavrov@omsu.ru

---

**МАТЕМАТИЧЕСКИЕ СТРУКТУРЫ и МОДЕЛИРОВАНИЕ**

В журнале публикуются статьи, в которых излагаются результаты исследований по фундаментальной и прикладной математике, теоретической физике и размышления, касающиеся окружающей нас природы и общества.

Публикуются также статьи по информационным технологиям, компьютерным наукам, защите информации, философии и истории математики.

Объекты исследования должны быть представлены в форме некоторых математических структур и моделей.

Журнал является реферируемым. Рефераты статей публикуются в «Реферативном журнале» и в журналах «Zentralblat für Mathematik» (Германия) и «Mathematical Reviews» (США).

Электронная версия журнала представлена в сети Интернет по адресам:

**http://msm.univer.omsk.su, http://msm.univer.omsk.ru**

Журнал издаётся на коммерческие средства факультета компьютерных наук Омского государственного университета.

Наш E-mail:

**lavrov@omsu.ru**

Подробную информацию можно найти на Web-сервере:

**http://msm.univer.omsk.su**

# СОДЕРЖАНИЕ

*Математические структуры и моделирование*
2011, вып. 23, с. 4–18

УДК 530.12

# ИЗМЕНЕНИЯ ТОПОЛОГИИ И ГЕОМЕТРИИ ПРОСТРАНСТВА, ПРИВОДЯЩИЕ К ОБРАЗОВАНИЮ КРОТОВОЙ НОРЫ

**А.К. Гуц**

В 3-мерном арифметическом пространстве $\mathbb{R}^3$ задаётся изменяющаяся с течением времени топология, от стандартной евклидовой до некомпактной неодносвязной, описывающей кротовую нору.

Классическое представление о физическом пространстве наделяет его таким фундаментальным топологическим свойством, как связность. Физическое пространство – суть связное 3-мерное многообразие – объединяется с временем в единое 4-мерное пространство-время.

Топология 3-пространства, а точнее, такие его топологические свойства, как связность и односвязность, как показано в этой работе, могут изменяться при скачках энергии естественного (взрывы) или искусственного происхождения.

При нарушении связности пространства рождаются либо ответвления в пространстве-времени, либо 4-мерные кротовые норы. Образовавшееся ответвление – это, по сути дела, параллельный мир.

В случае нарушения односвязности появляются 3-мерные кротовые норы в пространстве.

И 4-мерные, и 3-мерные кротовые норы могут использоваться как для переходов в Прошлое (машина времени), так и для сверхбыстрых по часам Земля сверхдальних космических перелётов [1].

## 1. Физика образования 4-мерных кротовых нор

Если в пространстве-времени не существует или недоступна естественная кротовая нора, то придётся создавать её искусственный аналог.

В качестве одного из способов можно рассмотреть образование 4-мерной кротовой норы, начало которой находится в настоящем, а конец – либо в историческом прошлом, либо в историческом будущем. Следует заметить, что пространство-время с 4-мерной ручкой уже не является односвязным, оставаясь связным. Поэтому существуют пространственно-подобные несвязные гиперповерхности. При этом процесс рождения 4-мерной кротовой норы можно


Омский государственный университет им. Ф.М. Достоевского
E-mail: guts@omsu.ru

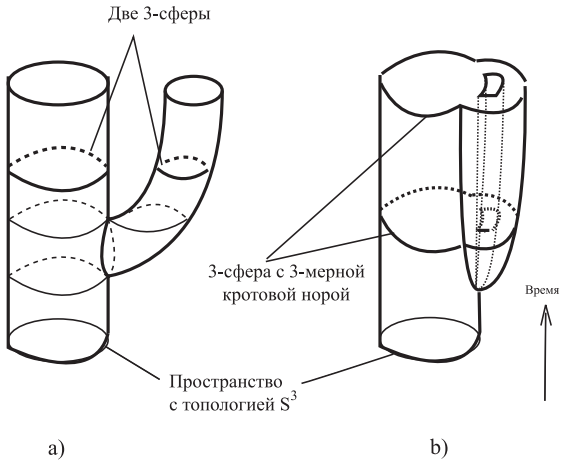


Рис. 1. a) рождение 4-мерной кротовой норы. Пространство с топологией 3-сферы теряет связность. Образуются два пространства, каждое из которых гомеоморфно 3-сфере. b) рождение 3-мерной кротовой норы в пространстве с топологией 3-сферы. Пространство теряет односвязность

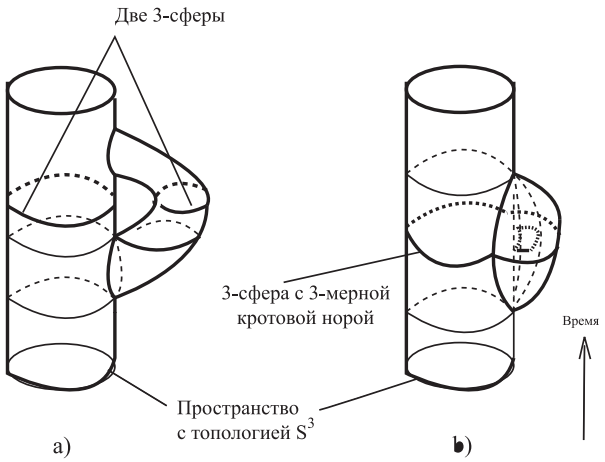


Рис. 2. a) рождение и исчезновение 4-мерной кротовой норы. Пространство с топологией 3-сферы теряет связность и вновь её обретает. b) рождение 3-мерной кротовой норы в пространстве с топологией 3-сферы. Пространство теряет односвязность, а затем опять становится односвязным

рассматривать как отрыв от 3-мерного пространства $M^3$ некоторой области $D_0 \subset M^3$.

Другими словами, образование 4-мерной кротовой норы означает нарушение связности пространственно-подобной гиперповерхности. Топологически (и геометрически) этот процесс может быть реализован стягиванием в точку границы отрываемой 3-мерной области $D_0$. В дальнейшем мы вклеим оторванную область в необходимую точку пространства-времени.

Следует отметить, что математически процедура склеивания двух несвязанных областей 3-мерного пространства представляется более сложным процессом, чем разрыв связной области на несвязные компоненты. Это вызвано тем, что при разрыве на две компоненты процесс стягивания границ в точку можно

обратить, так как эта точка является в действительности некоторой двумерной областью нулевой площади, полученной в результате непрерывной деформации. При склеивании двух несвязных компонент мы сначала выберем по точке в каждой области, а потом отождествим их. После этого точка, соединяющая склеенные части 3-мерной гиперповерхности, остаётся истинной точкой, в отличие от предыдущего случая. И мы не сможем так же просто растянуть точку в двумерную область.

### 1.1. Разрыв пространства

Разрыв пространства осуществим за счёт рассмотрения изменяющейся топологии и геометрии на одном и том же множестве $M^3$.

Можно построить вложение рассматриваемого множества $M^3$ в объемлющее 4-мерное пространство в виде семейства римановых 3-пространств, реализующих привычную картину разрыва на два несвязных куска. Эту картину оставляем как тренировку воображения читателя: её строгая математическая формализация не является сложной задачей.

### 1.2. Оценка скачка энергии, необходимого для разрыва пространства

Если теперь рассмотреть модель связного, но неодносвязного пространства-времени, то вполне можно обнаружить несвязные трёхмерные пространственно-подобные сечения. Более того, несвязное сечение $M_1^3$ может получиться из связного $M_0^3$ с помощью сферической перестройки [2], и, следовательно, связное и несвязное сечения можно рассматривать как начальное и конечное состояния некоторого геометродинамического процесса (лоренцев кобордизм [2]). В ходе этого процесса 3-геометрия претерпевает переход через некоторое критическое состояние $M_{1/2}^3$, которое отвечает нарушению связности пространственно-подобного сечения.

Было бы интересно выяснить [2], при каких условиях происходит нарушение связности пространственно-подобных сечений, или, если оставить в стороне конкретную дифференциально-топологическую модель, выяснить — возможно ли, что в ходе некоторого физического процесса трёхмерное пространство $M_0^3$ становится несвязным. Допуская вольность в словах, можно сказать, что нарушение связности означает отрывание области $D_0$ от $M_0^3$.

Рождение 4-мерной кротовой норы означает, что 3-мерный кусок $D_0$ отделяется, оставляет 3-мерное физическое пространство $M_0^3$.

Переход от $M_0^3$ к $M_1^3$ можно осуществить, стягивая в точку $\alpha^*$ границу $\partial D_0$ замкнутой области $D_0 \subset M_0^3$. Получается пространство $M_{1/2} = C_{1/2} \bigcup D_{1/2}$ , где $C_{1/2}$ и $D_{1/2}$ имеют одну общую точку $\alpha^*$ (результат стягивания $\partial D_0$) и являются связными гладкими многообразиями, диффеоморфными связными компонентами пространства $M_1^3$. Затем идёт отрыв $C_{1/2}$ от $D_{1/2}$; получаем $M_1^3$.

Геометрически (метрически) нарушение связности можно охарактеризовать как процесс уменьшения до нуля площади поверхности $\partial D_0$, ограничиваю-

щей отрывающуюся область $D_0$. Значит, связность пространства нарушается вследствие возмущения 3-метрики $\gamma_{\alpha\beta} \to \gamma_{\alpha\beta} + \delta\gamma_{\alpha\beta}$ $(\alpha, \beta = 1, 2, 3)$. Локальное возмущение 3-метрики ведёт к изменению кривизны 3-пространства. В рамках общей теории относительности 3-пространство рассматривается как пространственно-подобное сечение пространства-времени. Поэтому следует исходить из возмущения 4-метрики $g_{ik}$ $(i, k = 0, 1, 2, 3)$ пространства-времени, индуцирующего возмущение 3-метрики $\gamma_{\alpha\beta}$ 3-пространства. Согласно уравнениям Эйнштейна, исходной причиной возмущения метрики является появление дополнительного локального энергетического источника. Необходимые затраты энергии, влекущие нарушение связности 3-пространства, можно было бы легко подсчитать, если бы имелась формула, связывающая некоторую числовую характеристику связности пространства с кривизной этого пространства.

В случае замкнутого 3-пространства $M^3$ такой числовой характеристикой является нульмерное число Бетти $\beta_0(M^3)$ [3]. Необходимая же формула также имеется, правда, лишь для частного случая замкнутого ориентированного риманова 3-пространства $M^3$ с метрикой $\gamma_{\alpha\beta}$, допускающего регулярное единичное киллингово векторное поле $\xi$ [4]:

$$\frac{1}{2\pi l(\xi)} \int\limits_{M^3} \{K(\xi^{\perp}) + 3K(\xi)\} dv = 2\beta_0(M^3) - \beta_1(M^3) + d_0, \tag{1}$$

где $d_0 = 0$ или 1 в зависимости от чётности или нечётности одномерного числа Бетти $\beta_1(M^3)$; $K(\xi^{\perp})$ – значение римановой кривизны в плоскости, ортогональной $\xi$; $K(\xi)$ – значение римановой кривизны для любой плоскости, содержащей $\xi$ (отметим, что $K(\xi)$ не зависит от выбора плоскости); $dv$ — форма объёма; $l(\xi)$ – длина интегральной траектории поля $\xi$ (она постоянна).

Осуществим отрывание области $D_0$ следующим образом. На 3-многообразии $M_0^3$ зададим семейство римановых метрик $\gamma_{\alpha\beta}(t)$, $t \in [0, 1]$, удовлетворяющее условиям:

а) $\gamma_{\alpha\beta}(t)$ при $0 \leq t < 1/2$ $C^2$-гладкое тензорное поле, а при $t \geq 1/2$ оно имеет разрывы производных первого рода на границе $\partial D_0$ замкнутой области $D_0$;

б) (стягивание $\partial D_0$ в точку $\alpha^*$) площадь $\sigma_t$ границы $\partial D_0$, вычисленная в метрике $\gamma_{\alpha\beta}(t)$, стремится к нулю при $t \to 1/2 - 0$, или, иначе,

$$dv_t\big|_{\partial D_0} \underset{t \to \frac{1}{2} - 0}{\longrightarrow} 0$$

и

$$dv_t\big|_{\partial D_0} = 0 \quad \text{при} \quad t \geq 1/2,$$

где $dv_t$ — форма объёма в метрике $\gamma_{\alpha\beta}(t)$; $dv_s/dv_t \leq 1$ на $M_0^3$, $t < \frac{1}{2} < s$;

в) пространство $<M_0^3, \gamma_{\alpha\beta}(0)>$, т.е. $M_0^3$ с метрикой $\gamma_{\alpha\beta}(0)$ является связным $C^2$-гладким римановым многообразием, а $C_t \equiv (M_0 \setminus D_0) \bigcup \{\alpha^*\}$ и $D_t \equiv D_0 \bigcup \{\alpha^*\}$ с метрикой $\gamma_{\alpha\beta}(t)$, $t \geq 1/2$, и дополненные точкой $\alpha^*$, представляют собой $C^2$-гладкие связные римановы замкнутые многообразия;

г) $\partial\gamma_{\alpha\beta}/\partial n$, где $n$ — нормаль к пространству $<M_0^3, \gamma_{\alpha\beta}(t)>$, непрерывны;

д) $\gamma_{\alpha\beta}(t) = \gamma_{\alpha\beta}(0)$ вне окрестности $O_\varepsilon$ области $D_0$;

е) пространство $<M_0^3, \gamma_{\alpha\beta}(t)>$, $t > 1/2$ имеет неотрицательную кривизну;

ж) пространство $<M_0^3, \gamma_{\alpha\beta}(t)>$, $t \in [0,1]$ допускает регулярное единичное киллингово поле $\xi_t$.

Последнее предположение самое неприятное, так как в ходе отрыва $D_0$ от $M_0^3$ симметрия 3-пространства, по-видимому, может исчезнуть при приближении к критическому значению $t = 1/2$. Но, понимая это, мы вынуждены вводить условие «ж» для того, чтобы иметь право пользоваться формулой (1). Отметим, что на необходимость допустить симметрии как средство хоть как-то продвинуться в решении поставленной нами задачи указывал автор работы [2].

Индексом $t$ будем помечать объекты, относящиеся к пространству $<M_0^3, \gamma_{\alpha\beta}(t)>$.

Для простоты будем считать, что всегда $\beta_1 = 0$. Пространство $<M_0^3, \gamma_{\alpha\beta}(t)>$ при $t < 1/2$ связно, и поэтому

$$\int\limits_{M_0^3} f(\xi_t) dv_t = 4\pi l(\xi_t), \tag{2}$$

где

$$f(\xi_t) = K(\xi_t^\perp) + 3K(\xi_t).$$

При $s > 1/2$ пространство $<M_0^3, \gamma_{\alpha\beta}(s)>$ имеет уже две связные компоненты. Следовательно,

$$\int\limits_{C_s} f(\xi_s) dv_s = 4\pi l(\xi_s'), \qquad \int\limits_{D_s} f(\xi_s) dv_s = 4\pi l(\xi_s''), \tag{3}$$

где штрихи над $\xi_s$ различают поле $\xi_s$ на связных компонентах.

Из (2), (3) получаем

$$\int\limits_{O_\varepsilon} \{f(\xi_s) dv_s - f(\xi_t) dv_t\} = 4\pi \{l(\xi_s') + l(\xi_s'') - l(\xi_t)\}.$$

Естественно считать, что объем области $D_0$ мал по сравнению со всем пространством. Поэтому $l(\xi_s') \sim l(\xi_t)$, а $l(\xi_s'')$ по порядку величины совпадает с линейным размером $\lambda$ области $D_0$. Далее, в $O_\varepsilon$ для достаточно близких к $1/2$ значений $t$, $s$ $dv_s/dv_t \leq 1$ в силу «б». Но тогда благодаря условию «е» имеем

$$\int\limits_{O_\varepsilon} f(\xi_s) dv_t \geq \int\limits_{O_\varepsilon} f(\xi_s) \frac{dv_s}{dv_t} dv_t \sim 4\pi\lambda + \int\limits_{O_\varepsilon} f(\xi_t) dv_t,$$

т.е.

$$\int\limits_{O_\varepsilon} \delta f \cdot dv_t \sim 4\pi\lambda, \tag{4}$$

где $\delta f \equiv f(\xi_s) - f(\xi_t)$, точнее,

$$\delta f \equiv \lim_{s \to 1/2+0} f(\xi_s) - \lim_{t \to 1/2-0} f(\xi_t).$$

Вводя среднее значение величины g

$$\langle \mathrm{g} \rangle \ = \frac{1}{v_t(O_\varepsilon)} \int\limits_{O_\varepsilon} \mathrm{g} dv_t,$$

где $v_t(O_\varepsilon)$ – объем области $O_\varepsilon$ в метрике $\gamma_{\alpha\beta}(t)$, перепишем (4) в следующем виде:

$$\langle \delta f \rangle \cdot \ v_t(O_\varepsilon) \sim 4\pi\lambda. \tag{5}$$

Это соотношение говорит о том, что отрыв области $D_0$ сопровождается скачком кривизны 3-пространства. Так как для скалярной кривизны $R^{(3)}$ 3-пространства можно написать [5, с.140].

$$R_t^{(3)} = 2\{K(\xi_t^{\perp}) + 2K(\xi_t)\},$$

то следует предположить

$$\langle \delta R^{(3)} \rangle \ \sim \ \langle \delta f \rangle. \tag{6}$$

Из уравнений Эйнштейна имеем [6, с.157]

$$R_t^{(3)} + K_{2,t} = \frac{16\pi G}{c^4}\, \varepsilon(t), \tag{7}$$

где

$$K_{2,t} = (K_\alpha^{\cdot\alpha})^2 - K_{\alpha\beta}K^{\alpha\beta},$$

и $K_{\alpha\beta}$ – тензор внешней кривизны пространственного сечения; $\varepsilon(t)$ — плотность энергии.

Благодаря условию «г», инвариант $K_{2,t} = K_{2,t}(x)$, $x \in M_0$, $t \in [0,1]$ будет непрерывной функцией на $M_0^3 \times [0,1]$. Следовательно, если $\delta K_2 = K_{2,s} - K_{2,t}$, то

$$\langle \delta K_2 \rangle \ = [K_{2,s} - K_{2,t}]\big|_{x=x_0(t,s)} \underset{\substack{t\to\frac{1}{2}-0 \\ s\to\frac{1}{2}+0}}{\longrightarrow} 0.$$

Поэтому для некоторых $t_0 < 1/2$ и $1/2 < s_0$ величина $\langle \delta K_2 \rangle$ пренебрежимо мала, и тогда из (5), (6), (7) получаем

$$\langle \delta\varepsilon \rangle \sim \frac{c^4}{4\pi G}\, \frac{\lambda}{v_{t_0}(O_\varepsilon)}\,,$$

или можно написать

$$\boxed{\langle \delta\varepsilon \rangle \sim \frac{c^4}{4\pi G}\, \frac{1}{\sigma},} \tag{8}$$

где $\sigma$ — характерное сечение области $D_0$.

Требуемая оценка получена[1].

[1] В статье [7] оценка (3.32) выведена без предположения о компактности 3-пространства.

### 1.3. Учёт скачка внешней кривизны 3-пространства

Мы получили оценку (8) скачка энергии при условии г), означающего непрерывность внешней кривизны $K_{\alpha\beta}$ пространственно-подобной гиперповерхности. Величина $\langle\delta\varepsilon\rangle$ обратно пропорциональна площади характерного сечения $\sigma$ отрываемой области $D_0$. Для уменьшения скачка плотности энергии необходимо отказаться от непрерывного изменения внешней кривизны 3-мерного пространства в процессе нарушения его связности [8].

Отметим, что тензор $K_{\alpha\beta}$ задаётся соотношением

$$K_{\alpha\beta} = -e_\beta \nabla_\alpha n,$$

в котором базисные вектора $e_\beta$ на гиперповерхности $M_0^3$ мы выбрали совпадающими с соответствующими базисными векторами в пространстве-времени, а временную координату $x^0$ зададим таким образом, чтобы вектор нормали $n$ совпадал с $e_0$. Такая система координат в 4-мерном пространстве-времени будет синхронной, а система координат на гиперповерхности – гауссовой нормальной (если мы дополнительно будет полагать $g_{00} = 1$). В этом случае

$$K_{\alpha\beta} = \frac{1}{2}\frac{\partial\gamma_{\alpha\beta}}{\partial n} = \frac{1}{2}\frac{\partial\gamma_{\alpha\beta}}{\partial x^0}.$$

Предполагаем, что при $t = 1/2$ производная по времени от метрического тензора $\gamma_{\alpha\beta}$ имеет разрыв первого рода. Это означает разрыв первого рода компонент тензора внешней кривизны.

Тогда из уравнения (7) имеем:

$$\langle\delta R^{(3)}\rangle + \langle\delta K_2\rangle = \frac{16\pi G}{c^4}\langle\delta\varepsilon\rangle. \tag{9}$$

Из (5), (6) следует, что

$$\langle\delta R_t^{(3)}\rangle \sim \frac{4\pi}{\sigma}. \tag{10}$$

Таким образом, можно надеяться снизить затраты на скачок энергии для разрыва пространства, если принять, что скачок внешней кривизны равен

$$\langle\delta K_2\rangle \sim \frac{1}{\sigma}$$

или, что равносильно,

$$\left\langle\delta\left(\frac{\partial\gamma_{\alpha\beta}}{\partial x^0}\right)\right\rangle \sim \frac{1}{\sqrt{\sigma}}. \tag{11}$$

Отметим, что внешняя кривизна имеет значение не только в момент разрыва двух областей пространственно-подобной гиперповерхности, но и при стягивании границы области $D_0$ в точку. Внешняя кривизна, по своей сути, определяет характер вложения гиперповерхности в объемлющее пространство. Поэтому стягивание границы отрываемой области в точку $\alpha^*$, несомненно влекущее непрерывную деформацию гиперповерхности, соответствует изменению внешней кривизны пространства $M_t^3$.

## 2. Топологическое описание образования 4-мерной кротовой норы

До сих пор основные результаты о нарушении связности пространственноподобной гиперповерхности касались физической стороны вопроса. Давайте посмотрим на это явление с топологической точки зрения. Покажем, как реализуется разрыв некоторой области на две области.

Разрыв будем осуществлять за счёт рассмотрения семейства изменяющихся топологий на одном и том же множестве $M$.

Но прежде напомним полезное для наших целей определение топологии.

### 2.1. Топология и её задание

Если дано множество точек, то можно задать вопрос о форме, которую это множество имеет. В современной математике слово «форма» заменяется на слово «топология».

Топология (форма) множества $M$ зависит от того, как для каждой точки задаётся то, что называется её *окрестностями*. Окрестность $U_x$ точки $x$ – это перечень тех других точек множества $M$, которые объявляются *близкими* к точке $x$.

*Топология на множестве* $M$ – это семейство $\mathcal{T} = \{(U_x^\alpha)_{\alpha\in A_x} : x \in M\}$ окрестностей точек множества $M$, удовлетворяющих двум условиям:

a) для любых $x \in M$ и $\alpha \in A_x$ $x \in U_x^\alpha$;

b) для любых $U_x^\alpha, U_x^{\alpha'}$ существует $U_x^{\alpha''} \subset U_x^\alpha \cap U_x\alpha'$.

На одном и том же множестве $M$ можно задать разные топологии, переопределяя то, что считать близким к той или иной точке. В результате *форма* (топология) у одного и того же множества может меняться самым кардинальным образом. Например, множество может иметь форму отрезка прямой, а может быть парой отрезков. Как это может быть сделано, показано в § 2.2.

### 2.2. Нарушения связности отрезка

Для начала разорвём отрезок [0,1] на два: [0,1/2] и [1/2,1], при этом единственная на отрезке [0,1] точка $1/2$ раздваивается.

На отрезке [0,1] определим параметрическое семейство функций $f_t(x)$, $t \in [0,1]$, такое, что для любого $x \in [0,1]$

$$f_0(x) = 1, \qquad f_1(x) = 2|x - 1/2|,$$

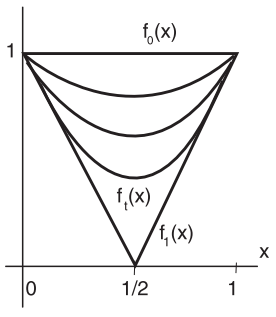


Рис. 3. Функции $f_t(x)$

а при $t \in (0,1)$ семейство функций $f_t$ представляет непрерывную деформацию функции $f_0$ в функцию $f_1$, причём все $f_t(x)$ $(t < 1)$ непрерывны вместе с первыми производными. Единственной функцией, производная которой имеет разрыв первого рода в точке

$x = 1/2$, является $f_1(x)$ (рис. 3). Более того, для любого $t$

$$f_t(1) = f_t(1) = 1.$$

Рассмотрим топологическое подпространство

$$\Gamma_t = \{(x, f_t(x), f_t'(x-0),\ (x, f_t(x), f_t'(x+0)\}$$

с индуцированной топологией трехмерного арифметического пространства $\mathbb{R}^3$, где $f_t(x \pm 0) = \lim\limits_{z \to x \pm 0} f_t(z)$. Две точки $(a, b, \alpha)$ и $(c, d, \beta)$ пространства $\Gamma_t$ назовём эквивалентными тогда и только тогда, когда

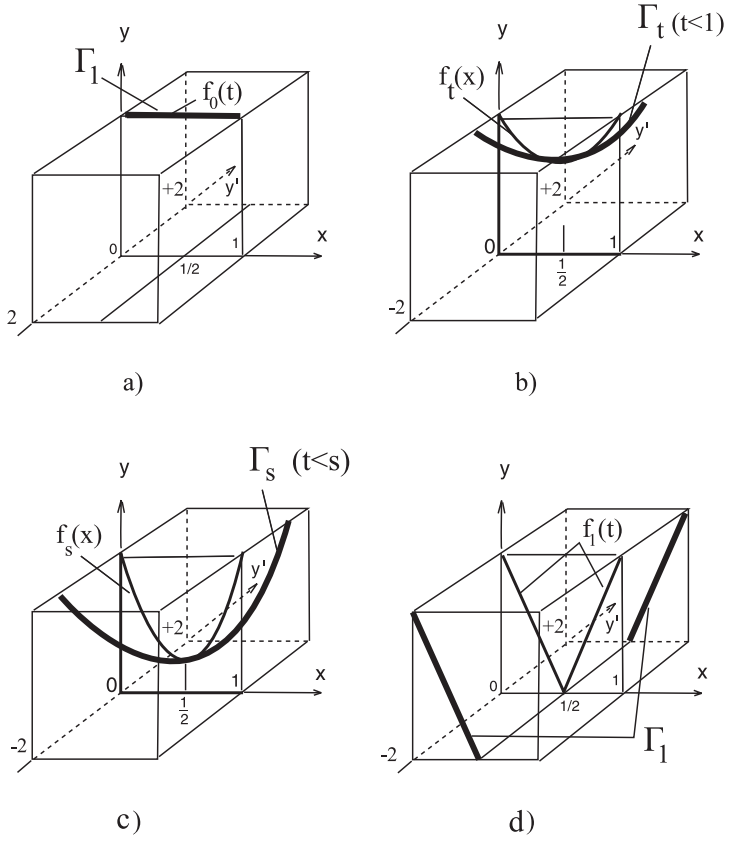


Рис. 4. a) пространство $\Gamma_0$ – отрезок; b) пространство $\Gamma_t$ при $0 \le t < 1$ – это кривая $x \to (x, f_t(x), f_t'(x))$, гомеоморфная отрезку [0,1]; c) пространство $\Gamma_s$ при $t < s < 1$ – это кривая $x \to (x, f_s(x), f_s'(x))$, гомеоморфная отрезку [0,1]; d) пространство $\Gamma_1$ (справа) – это два отрезка; $\Gamma_1$ несвязно и гомеоморфно паре отрезков $[0,1] \cap [2,3]$

1) $a = c$;
2) $b = d$;
3) $\alpha \equiv \lim\limits_{x \to a-0} f_t'(x) = \lim\limits_{x \to c+0} f_t'(x) \equiv \beta$.

Профакторизуем пространство $\Gamma_t$ по введённому отношению эквивалентности $\sim$. Получаем фактор-пространство $\Gamma_t/_\sim$.

Нетрудно увидеть, что это фактор-пространство при $t < 1$ гомеоморфно отрезку [0,1], а при $t = 1$ несвязному пространству $[0,1] \cup [2,3]$. Удобно последнее

несвязное пространство обозначить как $\Gamma_1/_\sim = [0,(1/2)_-] \cup [(1/2)_+,1]$. Итак, мы определили семейство топологических пространств $\{\langle \Gamma_t/_\sim, \mathcal{T}_t\rangle\}$, где $\mathcal{T}_t$ рассмотренная фактор-топология на $\Gamma_t/\sim$. При этом, если $t \neq 1$, то $\Gamma_t/_\sim \approx [0,1]$, а $\Gamma_1/_\sim = [0,(1/2)_-] \cup [(1/2)_+,1]$, и справедливы соотношения $[0,(1/2)_-] \approx [0,1]$ и $[(1/2)_+,1] \approx [0,1]$. Таким образом, мы рассмотрели разрыв отрезка на два с топологической точки зрения.

### 2.3. Нарушение связности для сфер $S^2$ и $S^3$

**Деление сферы** $S^2$. Разрыв сферы $S^2$ проведём по аналогии с представленными выше результатами. Введём линейное отображение $\mu(\theta) : [-\pi/2, +\pi/2] \to [0,1]$ такое, что $\mu(-\pi/2) = 0, \mu(0) = 1/2, \mu(\pi/2) = 1$. Далее, на сфере зададим семейство функций $\widetilde{f}_t(\theta,\varphi)$ (здесь $\theta \in [-\pi/2, +\pi/2]$, $\varphi \in [0, 2\pi]$) такое, что

1) $\widetilde{f}_t(\theta,\varphi) = \widetilde{f}_t(\theta)$;

2) $\widetilde{f}_t(\theta) = f_t(x), x = \mu(\theta)$, где $f_t(x)$ определена в предыдущем параграфе.

Рассмотрим множество пар

$$\Gamma_t = \{((\varphi,\theta), \widetilde{f}_t(\theta), \frac{d\widetilde{f}_t}{d\theta}(\theta - 0)),\ ((\varphi,\theta), \widetilde{f}_t(\theta), \frac{d\widetilde{f}_t}{d\theta}(\theta + 0))\}.$$

Тогда точки $A$ и $B$ множества $\Gamma_t$, такие что $A = ((a,b),\alpha,u)$ и $B = ((c,d),\beta,v)$, назовём эквивалентными тогда и только тогда, когда выполнены следующие условия:

во-первых,

1) $a = c$, $b = d$;

2) $\alpha = \beta$;

3) $v \equiv \lim\limits_{\theta\to b-0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta} = \lim\limits_{\theta\to b+0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta} \equiv u$;

во-вторых,

1) $b = d = 0$;

2) $\alpha = \beta$;

3) $v \equiv \lim\limits_{\theta\to 0-0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta} = \lim\limits_{\theta\to 0-0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta} \equiv u$ и $\lim\limits_{\theta\to 0-0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta} \neq \lim\limits_{\theta\to 0+0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta}$

или, в-третьих,

1) $b = d = 0$;

2) $\alpha = \beta$;

3) $v \equiv \lim\limits_{\theta\to 0+0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta} = \lim\limits_{\theta\to 0+0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta} \equiv u$ и $\lim\limits_{\theta\to 0-0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta} \neq \lim\limits_{\theta\to 0+0} \dfrac{d\widetilde{f}_t(\theta)}{d\theta}$.

Два последних условия эквивалентности отождествляют все точки экватора $\theta = 0$ – стягивают экватор в точку. Причём экватор раздваивается в зависимости от того $\theta \to 0 + 0$ (верхний берег экватора, рис. 5) или $\theta \to 0 - 0$ (нижний берег экватора, рис. 5). Поэтому при стягивании появляются две точки, и, следовательно, сфера делится на две разные сферы (рис. 5).

Таким образом, мы определили семейства топологических пространств

$$\{\langle \Gamma_t/_\sim, \mathcal{T}_t\rangle\}.$$

Нетрудно понять, что при $t \neq 1 \quad \langle \Gamma_t/_\sim, \mathcal{T}_t \rangle \approx S^2$, а при $t = 1 \quad \langle \Gamma_1/_\sim, \mathcal{T}_t \rangle \approx S^2 \cup S^2$, т.е. две сферы (рис. 5).

**Деление сферы** $S^3$. Разрыв осуществляется аналогично разрыву двумерной сферы $S^3$. На экваторе для стягивании экваториальной 2-сферы в точку углы $\varphi$ и $\theta$ берутся любыми, а третья угловая координата $\chi$ выполняет роль $\theta$.

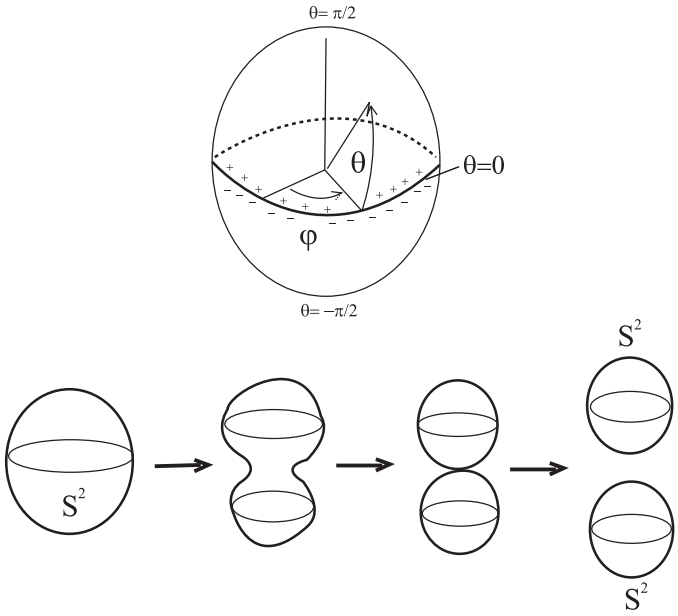


Рис. 5. Рождение 4-мерной кротовой норы в пространстве с топологией $S^2$. Пространство теряет связность; образуются две сферы

## 3. Топологическое описание образования 3-мерной кротовой норы

Аналогично процессу нарушения связности, означающего рождение 4-мерной кротовой норы за счёт изменения топологии на множестве $M$, описанному в предыдущем параграфе, можно таким же образом описать возникновение 3-мерной кротовой норы в пространстве.

### 3.1. Нарушение односвязности $\mathbb{R}^2$

Начнём в качестве примера с описания рождения 2-мерной кротовой норы у 2-мерной плоскости.

На отрезке [0,1] определим параметрическое семейство функций $h_t(x)$, $t \in [0,1]$ такое, что для любого $x \in [0,1]$ $h_0(x) = 1$, $h_t(0) = h_t(1) = 1$, и при $t \in (0,1)$ семейство функций $h_t$ представляет непрерывную деформацию функции $h_0$ в функцию $h_1$, причём все $h_t(x)$ непрерывны вместе с первыми производными. Единственной функцией, производная которой имеет разрывы первого в точках

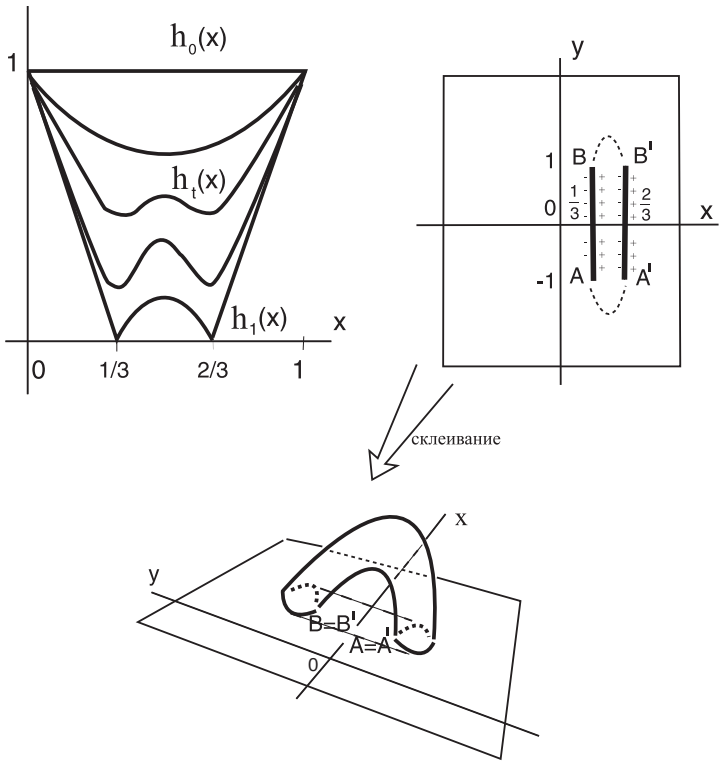


Рис. 6. Переход к неодносвязной поверхности за счёт разрезов по отрезкам $AB$, $A'B'$ и склеивания «левого берега» (минусы) отрезка $AB$ с «правым берегом» отрезка $A'B'$ (плюсы) и склеивания «правого берега» (плюсы) отрезка $AB$ с «левым берегом» отрезка $A'B'$ (плюсы). Получается плоскость с приклеенной 2-ручкой (2-мерная кротовая нора)

$x = 1/3$ и $x = 2/3$, является $h_1(x)$. Наконец, пусть (рис. 6)

$$\lim_{z\to 1/3-0} h_1'(z) = -1, \quad \lim_{z\to 1/3+0} h_1'(z) = +1$$

$$\lim_{z\to 2/3-0} h_1'(z) = -1, \quad \lim_{z\to 2/3+0} h_1'(z) = +1.$$

Рассмотрим топологическое подпространство

$$\Gamma_t = \{((x,y), h_t(x), h_t'(x-0)), \quad ((x',y'), h_t(x'), h_t'(x'+0))\}$$

с индуцированной топологией 4-мерного арифметического пространства $\mathbb{R}^4$, где $h_t'(x \pm 0) = \lim\limits_{z\to x\pm 0} h_t'(z)$. Две точки $((x,y), a, \alpha)$ и $((x',y'), b, \beta)$ пространства $\Gamma_t$ назовём эквивалентными, тогда и только тогда, когда, во-первых,

1) $(x,y) = (x',y')$;
2) $a = b$;
3) $\lim\limits_{z\to x-0} h_t'(z) = \lim\limits_{z\to x'+0} h_t'(z)$.

и, во-вторых, если

1) $x = 1/3, x' = 2/3$, $-1 \le y \le 1$;
2) $a = b$;
3) ( $\lim\limits_{z\to 1/3-0} h_t'(z) = -1$ & $\lim\limits_{z\to 2/3+0} h_t'(z) = +1$) или

$(\lim\limits_{z\to 1/3+0} h_t'(z) = +1 \;\&\; \lim\limits_{z\to 2/3-0} h_t'(z) = -1)$.

Профакторизуем пространство $\Gamma_t$ по введённому отношению эквивалентности $\sim$ . Получаем фактор-пространство $\Gamma_t/_\sim$. Нетрудно увидеть, что это фактор-пространство при $t < 1$ гомеоморфно плоскости $\mathbb{R}^2$, а при $t = 1$ неодносвязной поверхности с 2-мерной ручкой, которую физики называют 2-мерной кротовой норой.

Таким образом, мы рассмотрели процесс образования 2-мерной кротовой норы.

### 3.2. Нарушение односвязности $\mathbb{R}^3$

Рассмотрим параметрическое семейство функций $s_t(r)$, $t \in [0,1]$, $r \in [0,+\infty)$, такое, что для любого $r \in [0,+\infty)$

$$s_0(r) \equiv 1, \quad s_t(0) = 1, \quad \lim_{r\to+\infty} s_t(r) = 1$$

и при $r \in (0,+\infty)$ семейство функций $s_t$ представляет непрерывную деформацию функции $s_0$ в функцию $s_1$, причём все $s_t(x)$ непрерывны вместе с первыми производными. Единственной функцией, производная которой имеет разрыв первого в точке $r = 1$ является $s_1(x)$. Наконец, пусть (рис. 7)

$$\lim_{r\to 1-0} s_1'(r) = -1, \quad \lim_{r\to 1+0} s_1'(r) = +1.$$

Используем цилиндрическую систему координат в пространстве $\mathbb{R}^3$.

Пусть $Z = \{(r,\varphi,z) : r = 1, 0 \le \varphi \le 2\pi, 0 \le z \le 1\}$ – цилиндр (рис.8).

Рассмотрим топологическое подпространство $\Gamma_t = \{(r,\varphi,z), s_t(r), s_t'(r \pm 0)\}$ с индуцированной топологией пятимерного арифметического пространства $\mathbb{R}^5$, где $s_t'(r \pm 0) = \lim\limits_{\rho\to r\pm 0} s_t'(\rho)$.

Две точки $((r,\varphi,z), a, \alpha)$ и $((r',\varphi',z'), b, \beta)$ пространства $\Gamma_t$ назовём эквивалентными тогда и только тогда, когда, во-первых,

1) $(r,\varphi,z) = (r',\varphi',z')$;
2) $a = b$;
3) $\lim\limits_{\rho\to r-0} s_t'(\rho) = \lim\limits_{\rho\to r+0} s_t'(\rho)$;

и, во-вторых, если

1) $r = 1, \varphi' = \varphi + 2\pi, z' = z$ $(0 \le z, z' \le 1)$;
2) $a = b$;
3) $(\alpha = \beta = \lim\limits_{\rho\to 1-0} s_t'(\rho) = -1)$;

и, в-третьих,

1) $r = 1, \varphi' = \varphi + 2\pi, z' = z$ $(0 \le z, z' \le 1)$;
2) $a = b$;
3) $(\alpha = \beta = \lim\limits_{\rho\to 1+0} s_t'(\rho) = +1)$.

Профакторизуем пространство $\Gamma_t$ по введённому отношению эквивалентности $\sim$. Получаем фактор-пространство $\Gamma_t/_\sim$. Нетрудно увидеть, что это фактор-пространство при $t < 1$ гомеоморфно плоскости $\mathbb{R}^3$, а при $t = 1$ неодносвязному

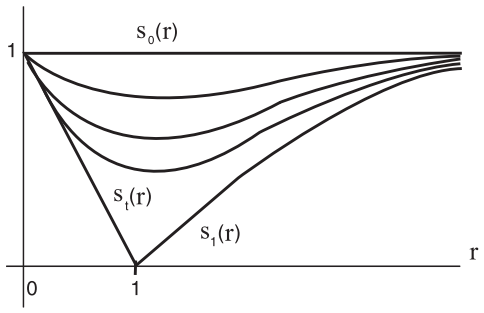


Рис. 7. Графики функций $s_t(r)$, $t \in [0,1]$

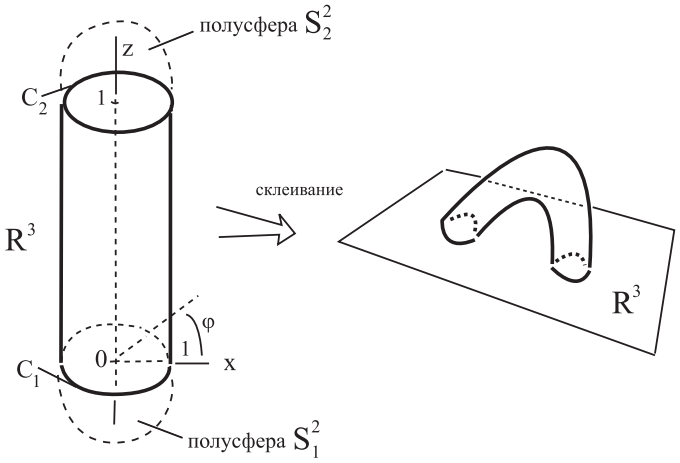


Рис. 8. Переход к неодносвязному 3-многообразию за счёт разреза по цилиндру $Z = \{(r,\varphi,z) : r = 1, 0 \le \varphi \le 2\pi, 0 \le z \le 1\}$ и склеивания «внешнего берега» по диаметрально противоположным точкам ($(1,\varphi,z)$ с $(1,\varphi+\pi,z)$) и «внутреннего берега» по диаметрально противоположным точкам. Окружности $C_1$ и $C_2$ каждая стягивается в точку; при этом полусферы $S_1^2$ и $S_2^2$ перестраиваются в сферы, к которым приклеен 3-мерный цилиндр $S^2 \times [0,1]$, получаемый из цилиндра $Z$ при факторизации по отношению эквивалентности $\sim$. Как результат, имеем 3-многообразие с приклеенной 3-ручкой (3-мерной кротовой норой)

некомпактному 3-многообразию с 3-мерной ручкой, которую физики называют 3-мерной кротовой норой.

**Изменение геометрии.** Зададим семейство римановых метрик,

$$dl_t^2 = A_t^2(r,z)[dr^2 + dz^2] + r^2[B_t(r,z)]^{-2}d\varphi^2,$$

отражающих изменение геометрии по мере изменения топологии, где функции $A_t, B_t$ выбираются так, что, во-первых, при $t < 1$ они $C^2$-гладкие, и, во-вторых, при $t = 1$ имеем: $\lim\limits_{r\to 1+0} A_t \ne \lim\limits_{r\to 1-0} A_t$, $\lim\limits_{r\to 1+0} B_t \ne \lim\limits_{r\to 1-0} B_t$ при $0 < z < 1$ и

$$A_t(1,c)[B_t(1,c)]^{-1} \to 0 \quad \text{при} \quad t \to 1$$

$$c = 0, 1.$$

Последнее условие — это геометрическое отражение условия стягивания окружностей $C_1, C_2$ в точку, при котором их длина должна стремиться к нулю.

## Литература


1. Гуц А.К. Космический корабль, разрушающий пространство? // Техника молодежи. 1983. №11. С.14-16.
2. Yodzis P. Lorentz cobordisms // Gen. Relat. and Gravit. 1973. V.4. P.299.
3. Спеньер Э. Алгебраическая топология. М. : Мир, 1971.
4. Reventos A. On the Gauss-Bonnet formula on the odd-dimensional manifolds // Tohoku Math. J. 1979. V.31, N.2. P.165-178.
5. Эйнзерхарт Л.П. Риманова геометрия. М. : ИЛ, 1948.
6. Мизнер Ч., Торн К., Уилер Дж. Гравитация. Т. 2. М. : Мир, 1977.
7. Гуц А.К. Нарушение связности физического пространства // Известия вузов. Физика. 1983. № 8. С. 3-6.
8. Палешева Е.В. Внешняя кривизна 3-мерного пространства и энергетические затраты, необходимые для образования 4-мерной кротовой норы // Математические структуры и моделирование. 2005. Вып. 15. С.89-96

*Математические структуры и моделирование*
2011, вып. 23, с. 19–30

УДК 519:577.4+502.7

# МОДЕЛЬ ЯРУСНО-МОЗАИЧНОГО ЛЕСА И МОДЕЛИРОВАНИЕ СУКЦЕССИИ

**А.К. Гуц, Е.О. Хлызов**

Предлагается математическая модель мозаичного многоярусного леса с целью описания серий сукцессии.

## Введение

В этой статье ставится задача построения математической динамической модели мозаичного многоярусного леса, с помощью которой можно было бы прогнозировать состояния лесной экосистемы, подверженной влиянию внешних управляющих факторов.

Модель основывается на четырёх управляющих внешних факторах, задающих среду экосистемы. Это влажность почвы $w$, мозаичность $m$, наличие конкуренции $k$ и антропогенное вмешательство $a$ в лесную экосистему (вырубка леса, пожары и т.д.). Модель может быть усложнена за счёт введения дополнительных управляющих внешних факторов, но при этом она становится менее наглядной и требует при её использовании уже гораздо более серьёзных математических знаний.

## 1. Понятие ярусности леса

Ярус – это неоднородность в вертикальном распределении фитомассы леса.

Дюрье выделяет крупные ярусы деревьев, кустарников, трав, напочвенного покрова, а в пределах этих ярусов – подъярусы:

– верхний, *первый ярус* $A$, образуют деревья. Его *подъярус* $A.1$ состоит из высоких деревьев; *подъярус* $A.2$ – деревья второй величины – рябина обыкновенная, черёмуха обыкновенная, ива козья, дикая яблоня.

– *второй ярус* $B$ состоит из кустарников, образующих подлесок, – лещина обыкновенная, жимолость лесная, крушина ломкая, бересклет европейский.

– *третий ярус* $C$ леса состоит из трав. Подъярус С.1 высоких трав – чистец лесной, бор развесистый, борцы; подъярус С.2 низких трав – сныть обыкновенная, осока волосистая, пролестник многолетний и др.

– *четвёртый ярус* $D$ – мхи, грибы, лишайники.


Омский государственный университет им. Ф.М. Достоевского
E-mail: guts@omsu.ru, hlyzov@gmail.com

Ярусное расположение растений связано с неодинаковой освещённостью. Количество света уменьшается от яруса к ярусу. Много света получают деревья первого яруса и очень мало – мхи и лишайники. В еловом лесу кустарники не растут: ветви елей задерживают очень много света, в таком лесу всегда сумрачно.

«Многоярусные биоценозы, представленные большим количеством видов растений, животных и микроорганизмов, связанных между собой разнообразными пищевыми и пространственными отношениями, называются сложными. Они наиболее устойчивы к неблагоприятным воздействиям. Исчезновение какого-либо вида существенно не отражается на судьбе таких биоценозов. В них происходит лишь незначительная перестройка организации, при которой популяции одного и даже нескольких видов могут заменяться экологически близкими видами, а стабильность сообщества определяется количественной регуляцией численности одних видов другими» [1].

## 2. Понятие мозаичности леса

Мозаичность – это неоднородность в горизонтальном (по площади) распределении фитомассы леса.

Мозаичность связана с неравномерным распределением деревьев в лесу. На рис. 1 представлены регулярное, случайное и контагиозное (пятнистое, мозаичное) распределения.

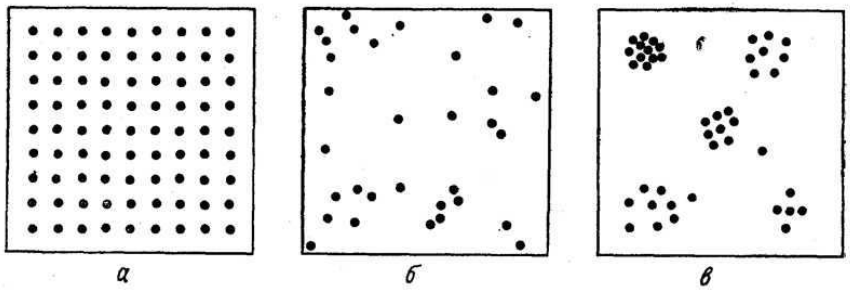


Рис. 1. Типы мозаичности леса: а) регулярное распределение, б) случайное распределение, в) контагиозное (пятнистое, мозаичное) распределение

Мозаичность в лесу может быть установлена с помощью, например, следующих двух способов измерения: коэффициента дисперсии и индекса размещения.

## 3. Шестиярусная модель леса

«Наиболее широко принятой в современной в экологии лесных сообществ является ярусно-мозаичная концепция леса как сложной системы» [2].

Лесной фитоценоз, его состояние, характеризуем с помощью такого понятия, как *первичная биологическая продуктивность леса*. Степень продуктивности фитоценоза в момент времени $t$ определяем как функцию и $x = x(t)$.

*Первичная биологическая продуктивность* характеризуется образованием биомассы (*первичной продукции*) в процессе фотосинтеза зелёными растениями (автотрофами), которые образуют первый трофический уровень экосистемы и служат началом всех цепей питания.

Продуктивность лесного биоценоза в момент времени $t$ будет определять продуктивность леса в следующий момент времени $t + dt$.

Иначе говоря,

$$x(t + dt) = x(t) + A(t, x)dt, \tag{1}$$

где $A(t, x)dt$ – величина, описывающая отклонения в продуктивности, произошедшие на отрезке времени $dt$.

Из (1) имеем то, что называется дифференциальным уравнением

$$\frac{dx}{dt} = A(t, x). \tag{2}$$

В основе продуктивного процесса растений лежит фотосинтез. Растения под воздействием солнечной энергии, поглощая листьями из атмосферы углекислый газ и корневой системой из почвы воду, создают органическое вещество. Недостаток влаги $(-w) > 0$ является фактором, не способствующим благополучию леса.

В таком случае следует в правую часть дифференциального уравнения (2) добавить член $(-(-w))$:

$$\frac{dx}{dt} = A(t, x) - (-w). \tag{3}$$

В самом простом случае можно принять, что $A(t, x) = k_1 x$, т.е.

$$\frac{dx}{dt} = k_1 x + w. \tag{4}$$

Коэффициент $k_1$ можно посчитать постоянным. Но тогда доброкачественность леса будет нарастать как геометрическая прогрессия, и это делает бессмысленной нашу модель. Поэтому начнём её усложнять.

Коэффициент $k_1$ отвечает за «прирост доброкачественности леса». Учтём, что для здорового леса обязательной чертой является наличие ярусности.

Выделим шесть ярусов леса: два яруса деревьев, один ярус кустарников, два яруса трав, один ярус напочвенного покрова (см. § 1).

«Каждый ярус, входящий в состав фитоценоза, оказывает влияние на другие ярусы и в свою очередь подвергается их влиянию. Поэтому фитоценоз необходимо рассматривать как нечто целое, а ярусы фитоценоза – как его структурные части, которые в некоторых случаях могут быть относительно самостоятельными» [1].

Взаимовлияние ярусов должно дать вклад в доброкачественность леса вида

$$\alpha_1 x \cdot \alpha_2 x \cdot \alpha_3 x \cdot \alpha_4 x \alpha_5 x \alpha_6 = \alpha x^6,$$

$$\alpha = \alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \alpha_6. \tag{5}$$

Каждый коэффициент $\alpha_i > 0$ характеризует степень участия $i$-го яруса во взаимодействии ярусов. Доля фитомассы $i$-го яруса характеризуется величиной $\alpha_i x$. Взаимовлияние $i$-го яруса и $j$-го яруса – это произведение $\alpha_i x \cdot \alpha_j x$.

Если $\alpha_i \to 0$, то мы констатируем отсутствие $i$-го яруса, означающее меньшую ярусность фитоценоза и, следовательно, его меньшую степень биоразнообразия, меньшую устойчивость к неблагоприятным воздействиям.

Примем, что

$$k_1 = \alpha x^6 - p(x), \tag{6}$$

где $\alpha = \alpha_1\alpha_2\alpha_3\alpha_4 = const$ – «сила» взаимодействия четырёх ярусов леса, а величина $p(x)$ – это то, что снижает продуктивность леса.

Тогда следует принять, что

$$p(x) = \underbrace{k_2}_{\text{вырубка леса, пожары}} + \underbrace{k_3x^2}_{\text{конкуренция}} - \underbrace{k_4x,}_{\text{мозаичность}} \tag{7}$$

где квадратичный член $k_3x^2$ характеризует конкуренцию растений в лесу (часть фитоцеоза – $\beta_1 x$ влияет на другую его часть – $\beta_2 x$. Взаимовлияние конкурирующих растений есть произведение $\beta_1 x \cdot \beta_2 x = k_3x^2$). Мозаичность леса пропорциональна, на наш взгляд, всей фитомассе $x$ леса – отсюда член $k_4x$. Коэффициент $k_4 > 0$ характеризует степень влияния горизонтальной структуры (мозаичности) на динамику прироста фитомассы.

Объединяя уравнения (4) – (7), получаем следующую *модель лесной экосистемы*:

$$\frac{dx}{dt} = \alpha x^7 - k_2x - k_3x^3 + k_4x^2 + w =$$

$$= \frac{\partial}{\partial x}\left\{\frac{\alpha}{8}x^8 + \frac{-k_3}{4}x^4 + \frac{k_4}{3}x^3 + \frac{-k_2}{2}x^2 + wx\right\} \tag{8}$$

или

$$\frac{dx}{dt} = \frac{\partial}{\partial x}V(x,k,m,a,w), \tag{9}$$

где

$$V(x,k,m,a,w) = \frac{\alpha}{8}x^8 + kx^4 + mx^3 + ax^2 + wx, \tag{10}$$

$$k = -\frac{k_3}{4}, \quad m = \frac{k_4}{3}, \quad a = -\frac{k_2}{2}.$$

Сопоставляем лесному фитоценозу потенциальную функцию

$$V(x,k,m,a,w) =$$

$$= \frac{\alpha}{8}x^8 + \underbrace{kx^4}_{\substack{\text{наличие}\\ \text{конкуренции}}} + \underbrace{mx^3}_{} + \underbrace{ax^2}_{\substack{\text{вырубка леса,}\\ \text{пожары,}\\ \text{ураган,}\\ \text{разлив нефти}}} + \underbrace{wx}_{\substack{\text{влажность}\\ \text{почвы}}}.$$

Это частный случай катастрофы типа «звезда» [4]:

$$V(x,p,q,k,m,a,w)=\frac{\alpha}{8}x^8+px^6+qx^5+kx^4+mx^3+ax^2+wx. \qquad (11)$$

Катастрофа «звезда» описывает, вообще-то говоря, семь равновесных состояний сразу, из которых не более четырёх являются устойчивыми (наличие сразу четырёх локальных минимумов у функции $V$).

Таким образом, имеем следующую модель шестиярусного лесного фитоценоза:

$$\frac{dx}{dt}=\frac{\partial}{\partial x}V(x,k,m,a,w), \qquad (12)$$

где

$$V(x,k,m,a,w)=\frac{\alpha}{8}x^8+kx^4+mx^3+ax^2+wx. \qquad (13)$$

Отметим, что продуктивность характеризуется неравенством $x>0$, наличие конкуренции неравенством $k<0$, действенность оконной динамики неравенством $m>0$, вырубка лесов, пожары неравенством $a<0$, недостаток влаги неравенством $w<0$.

При моделировании лесных экологических катастроф с помощью построенной модели важно дать ответ на следующий вопрос: если, например, при пожаре сгорели, скажем, два верхних яруса леса, то можно ли считать, что допустимо дальнейшее использование 6-ярусной модели (12) – (13)? Можно, особенно когда мы ограничиваемся качественным описанием сукцессии. Действительно, при выгоревших ярусах 1,2 мы имеем ситуацию, когда $\alpha_1\to 0$ и $\alpha_2\to 0$. Следовательно, уменьшается сила взаимодействия ярусов $k_0$, но мы имеем ту же потенциальную функцию (13), но с другим значением $\alpha$. Для качественного анализа изменение значения величины $\alpha$ не играет существенной роли.

## 4. Равновесия биоценоза; сукцессия как последовательная смена равновесий

*Сообщество* – это группа организмов различных видов, проживающих на общей территории и взаимодействующих между собой.

Конкретное значение продуктивности лесной экосистемы, находящейся в состоянии *равновесия*, будем интерпретировать как конкретное растительное сообщество. Смена равновесия, сопровождающаяся изменением значения продуктивности лесной экосистемы, – это смена одних сообществ другими (динамика растительного покрова по Сукачёву).

Равновесия лесного биоценоза, описываемого уравнением (12) – (13), находятся как решения $x=x(k,m,a,w)$ уравнения

$$\frac{\partial V(x,k,m,a,w)}{\partial x}=0. \qquad (14)$$

Лесную экосистему можно вывести из состояния равновесия многими способами: пожаром, наводнением или засухой. После такого нарушения равновесия

новая экосистема сама себя восстанавливает, и этот процесс носит регулярный характер, называется *сукцессией* и повторяется в самых разных ситуациях.

*Сукцессия* – это последовательный ряд смены серийных (временно существующих) растительных сообществ на конкретном местообитании после выведения конкретной экосистемы из состояния равновесия [1].

Можно также сказать, что сукцессия, как процесс, представляет собой последовательную (необратимую) смену биоценозов, преемственно возникающих на одной и той же территории в результате влияния природных или антропогенных факторов.

Различают множество форм сукцессии: фитогенная, зоогенная, ландшафтная, антропогенная, пирогенная, катастрофическая и др.

Каждое сообщество, промежуточное в серии, называемое *стадией*, существует достаточно долго и слабо меняется во времени. **Поэтому стадию можно также считать равновесным состоянием**.

Равновесие в природе на самом деле зависит от окружающей среды, а среда эта постоянно подвержена изменениям. Пожары, наводнения, колебания количества атмосферных осадков оказывают влияние на среду, в которой произрастает лес. И растения, конечно же, не могут не реагировать на эти изменения. Получается, что экосистема все время пытается либо сохранить равновесие, либо попасть в новое равновесие. Вмешательство человека – всего лишь ещё один способ изменить окружающую среду и таким образом повлиять на направление развития экосистемы.

Смена сообщества происходит под влиянием факторов, которые связаны с экосистемой и являются её характеристиками.

На роль таких факторов могут претендовать такие характеристики фитоценоза, как внутривидовая и межвидовая конкуренция, мозаичность, оконная динамика. Изменения этих факторов отражают в себе микроэволюцию фитосреды, т.е. накопление изменений фитосреды, сначала благоприятных для фитоценоза, а затем неблагоприятных, и, значит, характеризуют наступающую смену сообщества, *смену равновесия*.

Следовательно, сукцессия может моделироваться в рамках математической теории катастроф, созданная для описания резких изменений, смен, т.е. катастроф равновесных состояний. Катастрофы при изменении одних факторов могут интерпретироваться как изменение климаксного сообщества, а других, при фиксированных первых, как смена сообщества в серии.

При смене равновесия в нашей модели меняется значение продуктивности леса $x(t)$. Она нами определена так, что характеризует разные её значения, соответствующие разным лесным сообществам, т.е. разным стадиям сукцессии. Смена стадии – это смена равновесия, а в рамках теории катастроф Тома – это бифуркация, означающая то, что в физике называется фазовым переходом. Следовательно, смена равновесия, смена стадии может описываться с привлечением аппарата теории фазовых переходов. Такой подход применён в работе А.С. Исаева и др. [3]. Поскольку теория катастроф Тома включает в себя теорию фазовых переходов [5, 6], то наш подход является более общим, допускающим самые различные обобщения.

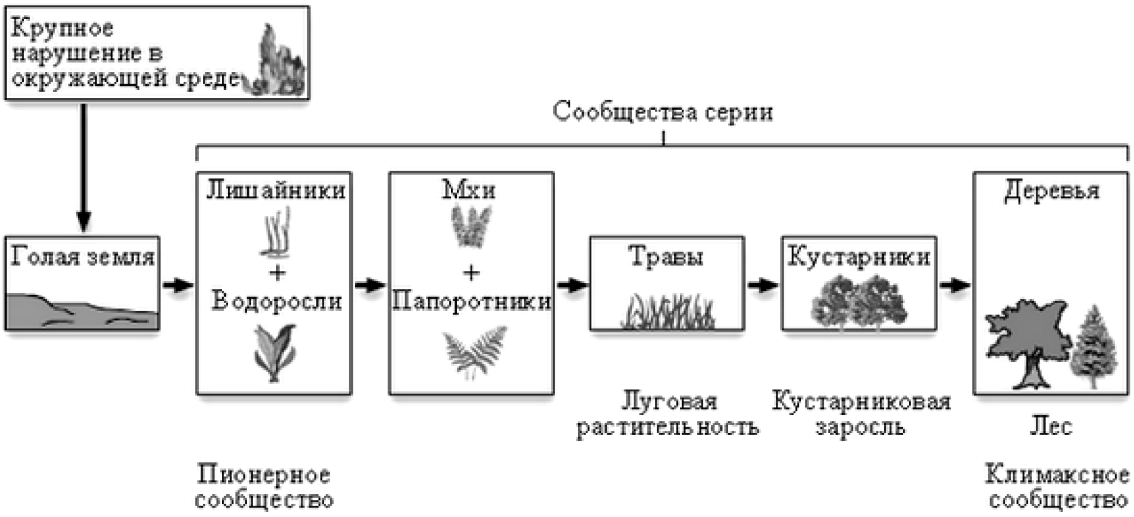


Рис. 2. Схема сукцессии [7]

Темпы сукцессий неодинаковы. На участке, лишённом растительного покрова, первые стадии сменяют друг друга через год несколько лет, а далее процесс смен замедляется, и более позднее стадии восстановления или формирования растительного покрова занимают десятилетия, а затем и столетия [1].

«Время смены одного сообщества другим сильно различается. Типичная последовательность сукцессий, приводящих к появлению дубрав или сосновых лесов в средней полосе, занимает около 200 лет; при этом скорости ранних сукцессий гораздо выше, чем скорости поздних.

В сходных условиях развиваются сходные сукцессии. Факторами, определяющими состав климаксного сообщества, могут быть климат, рельеф, дренаж почвы и т.д.

Нередко результатом сукцессии на таких территориях является восстановление исходного биогеоценоза. Применительно к лесным сообществам такие процессы получили название *демутация*» [7].

«При этом конкретная экосистема возвращается в своё исходное состояние и пребывает в нем до тех пор, пока не изменятся климат, рельеф, гидрологический режим, пока вновь не пройдёт пожар или не случится какая-то другая катастрофа. И вновь начнётся новая сукцессия, которая либо приведёт к восстановлению исходного сообщества, либо нет. В результате сукцессии на конкретном местообитании восстанавливается исходное растительное сообщество, называемое геоботаниками климаксовым, или коренным. Коренное сообщество растений устойчиво и в данных климатических условиях не изменяется» [1].

Если внешние факторы $k, m, a, w$, изменяясь в 4-мерном пространстве с осями $k, m, a, w$, пересекают так называемое *бифуркационное множество* $B_V$, то происходит резкая смена равновесия экосистемы, происходит то, что математики называют *бифуркацией* (катастрофой). Это соответствует экологической катастрофе, постигшей лесной биоценоз. Продолжающиеся изменения факторов – это новые бифуркции, новые переходы к новым равновесиям, которые

рассматриваем уже как серийные (временно существующие) растительные сообщества. Возврат факторов к исходным значениям – это создание условий к полному восстановлению биоценза.

Бифуркационное множество $B_V$ находят, исключая $x$ и решая систему уравнений:

$$\frac{\partial V(x,u_1,...,u_n)}{\partial x}=0, \quad \frac{\partial^2 V(x,u_1,...,u_n)}{\partial x^2}=0.$$

## 5. Смены климаксных сообществ

Последнее сообщество серии (цепи), стабильное и находящееся в равновесии с окружающей средой, называется *климаксным сообществом*. Дальнейшее изменение климаксного сообщества возможно только при изменении окружающих условий.

Мы видим, что климаксное сообщество – это *равновесное состояние* экосистемы, которое соответствует набору конкретных значений внешних факторов (влажность почвы, климат, антропогенные воздействия).

Смена климаксных сообществ происходит при действии экзогенных факторов, таких как пожары, разливы нефти, засуха и т.д.

В рамках нашей модели для описания смены климаксных сообществ мы должны рассмотреть проекции бифуркационного множества на плоскость «пожары (вырубка) $a$ – влажность $w$» или на плоскость «атмосфера $m$ – пожары (вырубка) $a$».

### 5.1. Бифуркации в плоскости «пожары (вырубка) $a$ – влажность $w$»

Бифуркации в плоскости пожары (вырубка) $a$ – влажность $w$ при фиксированных $p=q=0$, $m=0$, $k=-20$ (сильная конкуренция) даны на рис. 3.

Взятые точки: $(a,w)=(-7,0);(20,0);(60,0);(30,-11);(30,11);(24,-18);(24,18)$.

### 5.2. Бифуркации в плоскости «атмосфера $m$ – пожары (вырубка) $a$»

Фактор $m$ можно трактовать как состояние атмосферы, т.е. как экзогенный фактор, существенно влияющий на состояние лесной экосистемы.

Бифуркации в плоскости атмосфера $m$ – пожары (вырубка) $a$ дана при фиксированных $p=q=0$; $w=0$, $k=-20$ на рис. 4.

Взятые точки: $(m,a)=(-10,0);(10,0);(70,0);(35,-21);(35,21)$.

## 6. Смены стадий

Смены стадий происходят под воздействием эндогенных факторов, таких как конкуренция и мозаичность.

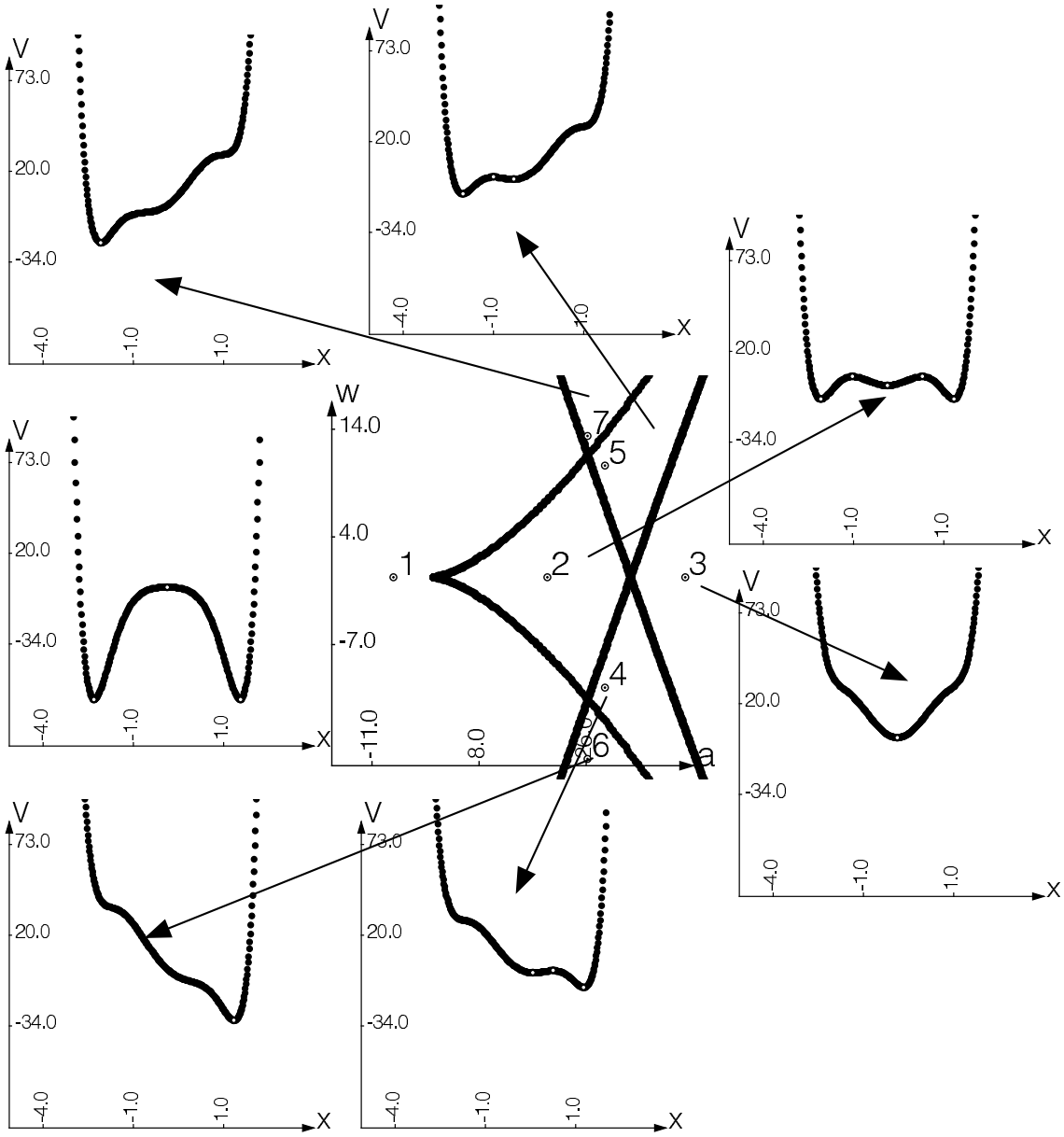


Рис. 3. Бифуркационное множество для 6-ярусного леса в плоскости «пожары (вырубка) $a$ – влажность $w$» в случае $m = 0$ и $k = -20$

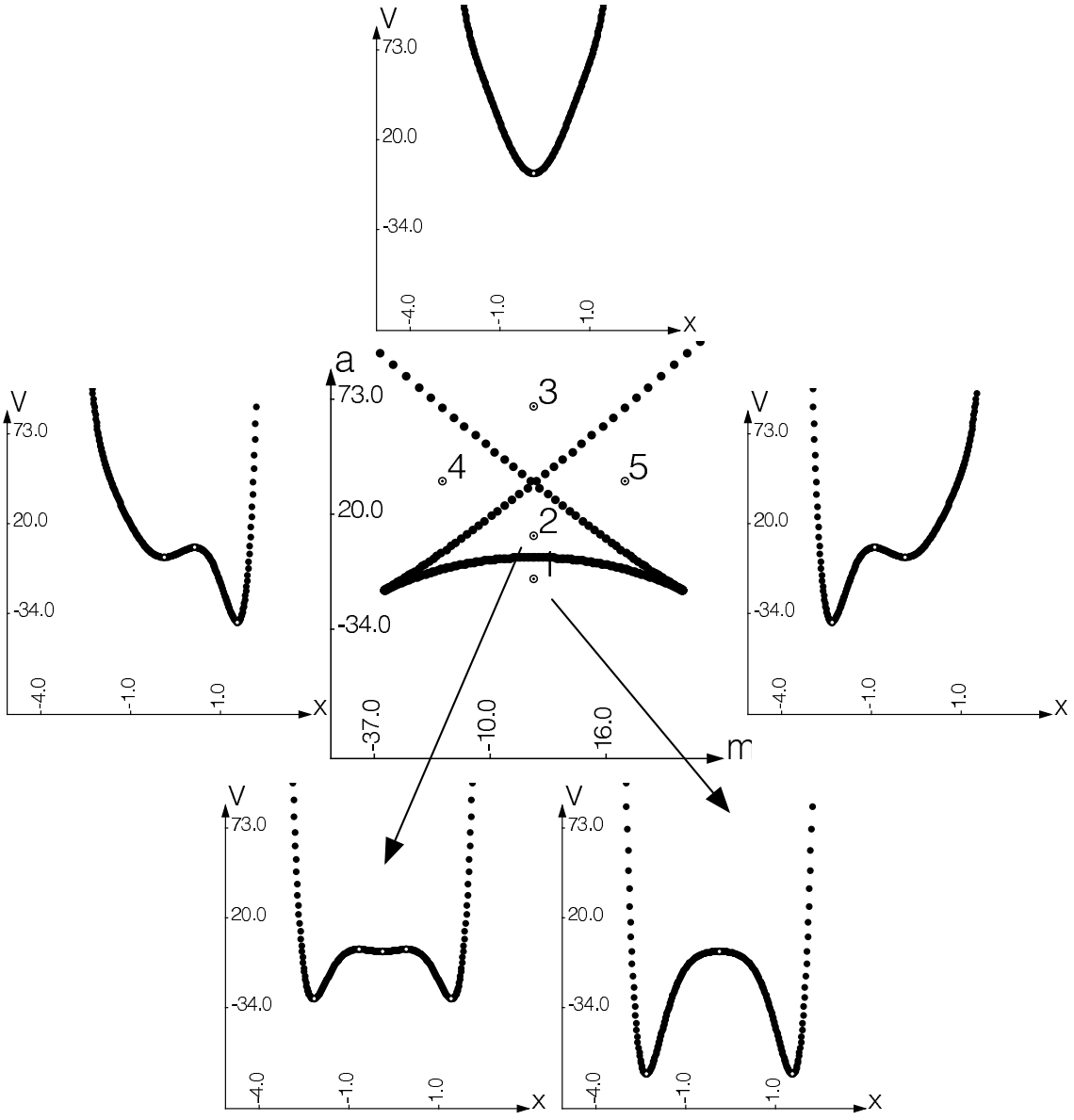


Рис. 4. Бифуркационное множество для 6-ярусного леса в плоскости «атмосфера $m$ – пожары (вырубка) $a$» в случае $w = 0$, $k = -20$

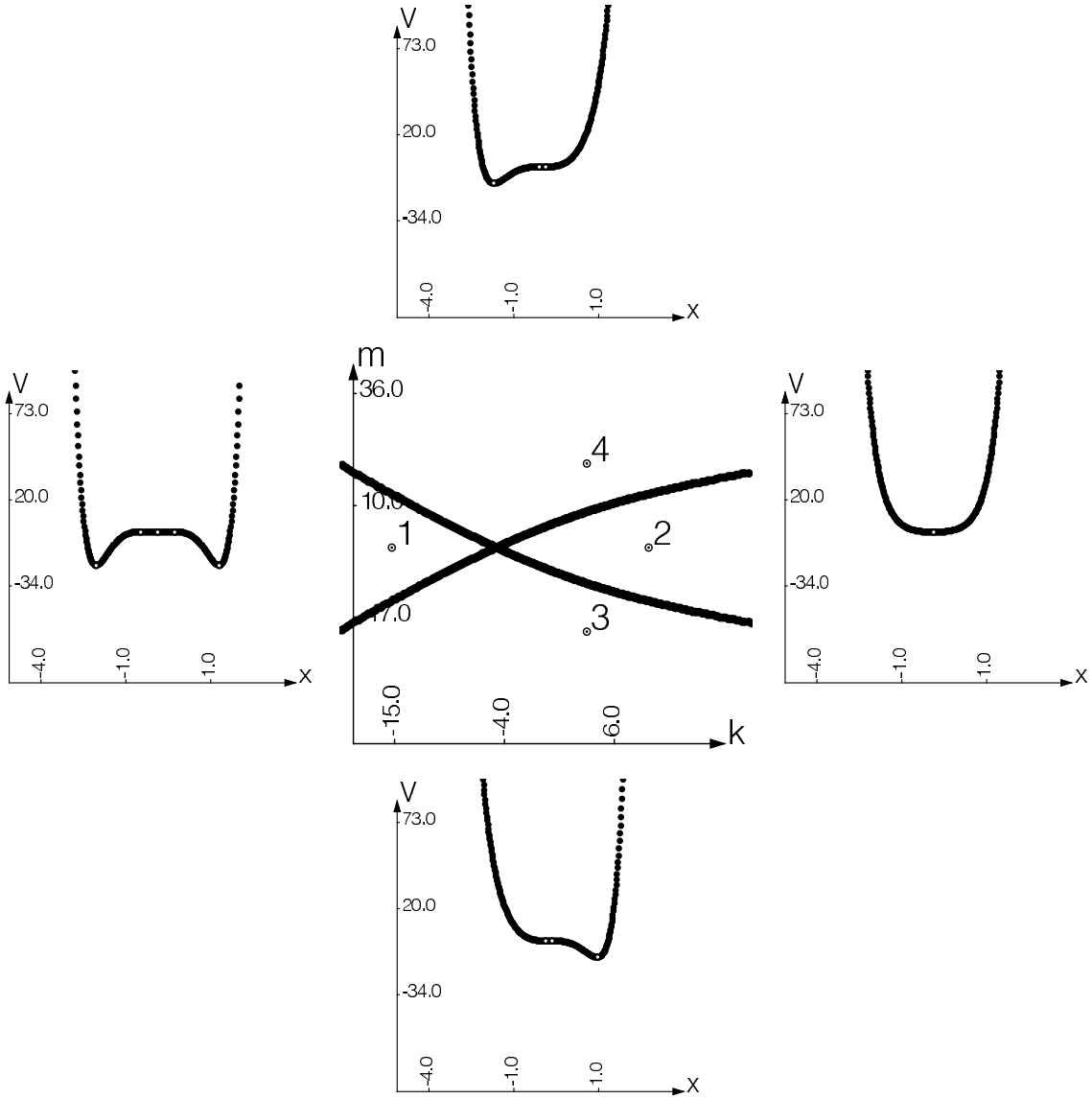


Рис. 5. Бифуркационное множество для 6-ярусного леса в плоскости «конкуренция $k$ – мозаичность $m$» в случае $w = 0$, $a = 20$.

В рамках нашей модели таковыми факторами являются бифуркации в плоскости «конкуренция $k$ – мозаичность $m$» дана при фиксированных $p = q = 0$; $w = 0$, $a = 20$ на рис. 5. Взятые точки: $(k, m) = (-20, 0); (1, 0); (0, 10); (0, -10)$.

## Литература


1. Москалюк Т.А. Курс лекций по биогеоценологии. URL: `http://www.botsad.ru/p_papers.htm` (дата обращения: 12.05.2009).
2. Ризниченко Г.Ю. Экология математическая. URL: `http://www.library.biophys.msu.ru/MathMod/EM.HTML#e9` (дата обращения: 16.05.09)
3. Исаев А.С., Суховольский В.Г., Бузыкин А.И., Овчинникова Т.М. Сукцессионные процессы в лесных сообществах: модели фазовых переходов // Хвойные бореальной зоны. 2008. Т.25, № 1-2. С.9-15.
4. Woodcock A., Poston T. A geometrical study of the elementary catastrophes // Lectures Notes in Math. № 373. Springer, 1974. 257 p.
5. Постон Т., Стюарт И. Теория катастроф и её приложения. М. : Мир, 1980.
6. Гилмор Р. Прикладная теория катастроф. М. : Мир, 1984.
7. Динамика природных сообществ // Биология. Электронный учебник. URL: `http://www.ebio.ru/eko07.html` (дата обращения: 16.05.09).

*Математические структуры и моделирование*
2011, вып. 23, с. 31–37

УДК 004.93'12:621.398

# РАСПОЗНАВАНИЕ ЛИЧНОСТИ ПО ПОХОДКЕ НА ОСНОВЕ WAVELET-ПАРАМЕТРИЗАЦИИ ПОКАЗАНИЙ АКСЕЛЕРОМЕТРОВ

**А.Г. Казанцева, Д.Н. Лавров**

В этой статье предлагается алгоритм распознавания личности по походке на основе wavelet-параметризации показаний акселерометров. Для сбора данных были использованы акселерометры, являющиеся частью устройства SunSPOT. Данные подвергаются предварительной обработке и дальнейшей параметризации на основе вейвлет-декомпозиции. Идентификация походки осуществляется с помощью предварительно обученной искусственной нейронной сети. Компьютерное моделирование даёт точность идентификации около 90%.

## Введение

Биометрические методы распознавания предполагают систему распознавания людей по одной или более физических или поведенческих черт. Биометрические данные можно разделить на два основных класса:

- Физиологические — относятся к форме тела. В качестве примера можно привести: отпечатки пальцев, распознавание лица, ДНК, ладонь руки, сетчатка глаза, запах.
- Поведенческие связаны с поведением человека. Например, походка и голос [1].

Чаще всего в мобильных устройствах для биометрической идентификации используются отпечатки пальцев. Этот метод более удобен, чем пароли, но тем не менее требует от пользователя активных действий.

Также были предложены методы идентификации на основе голоса, подписи и походки [2–4]. Однако идентификация по голосу или подписи тоже в той или иной мере требует вмешательства пользователя, в то время как идентификация по походке может быть абсолютно прозрачной для него.

В настоящее время портативные устройства, такие как мобильные телефоны, смартфоны, коммуникаторы, карманные персональные компьютеры и т.д., получили широкое распространение. Большинство из них оснащается акселе-


Омский государственный университет им. Ф.М. Достоевского
E-mail: dma666@bk.ru, lavrov@omsu.ru

рометрами, которые и позволяют замерять необходимые для идентификации параметры.

При расследовании различного рода преступлений и ведении оперативно-розыскных мероприятий возможны ситуации, когда необходимо идентифицировать владельца смартфона или коммуникатора. Одним из возможных способов идентификации личности на расстоянии в таких ситуациях также является идентификация по походке.

Целью данной работы является разработка и реализация метода идентификации личности по походке с параметризацией на основе дискретного вейвлет-преобразования и искусственной нейронной сетью (ИНС) в качестве классификатора.

Для достижения этой цели в работе поставлены следующие задачи:
- разработать метод параметризации на основе дискретного вейвлет-преобразования;
- эмпирически подобрать вейвлет, дающий лучшее распознавание;
- выбрать тип нейросети, алгоритм её обучения и другие параметры;
- обучить нейросеть;
- оценить результаты.

Проект будет реализован на языке программирования Java, так как этот язык является объектно-ориентированным, кроссплатформенным, обладает гибкой системой безопасности [5], а также прост в использовании.

Поскольку невозможно, не проводя эксперимента, предугадать, какой вейвлет даст лучшие результаты для поставленной задачи, необходима библиотека вейвлет-преобразований, поддерживающая широкий спектр различных вейвлетов.

Так как ни одна из известных библиотек на языке Java не реализует достаточное количество вейвлетов, необходимо разработать собственную библиотеку вейвлет-преобразований. Эта библиотека должна будет поддерживать основные семейства дискретных вейвлетов, такие как вейвлеты Хаара, Добеши, симлеты, койфлеты и биортогональные вейвлеты; быть легко расширяемой и удобной в использовании.

В качестве библиотеки нейросетей будет использоваться библиотека Encog, так как она поддерживает большое количество типов нейросетей, алгоритмов обучения [6], а также обладает более высокой производительностью, чем другие библиотеки [7].

## 1. Применение ИНС для идентификации личности

### 1.1. Исходные данные

Телеметрические данные снимались при помощи устройства SunSPOT (Sun Small Programmable Object Technology). SunSPOT — это беспроводное сенсорное устройство, оборудованное, среди прочего, трёхмерным акселерометром LIS3L02AQ и поддерживающее беспроводной протокол передачи данных IEEE 802.15.4 [8], что позволяет производить замеры на расстоянии до 10 метров от

базовой станции [9].

В качестве сигнала для последующей параметризации был выбран модуль ускорения $|a| = \sqrt{a_x^2 + a_y^2 + a_z^2}$, где $a_x$,$a_y$,$a_z$ - ускорения по осям $x$, $y$ и $z$ соответственно [10]. Этот выбор обусловлен тем, что модуль ускорения не зависит от ориентации датчика в пространстве. Таким образом, положение устройства не будет вносить погрешность в результаты измерений.

В ходе эксперимента были произведены измерения телеметрических параметров походки группы из трёх человек (группа 1), по 20 замеров на человека. Также были произведены измерения параметров походки группы из 22 человек (группа 2) для оценки вероятности ошибок второго рода, по одному замеру на человека. Показания акселерометра снимались с частотой 100 Hz. Во время измерений SunSPOT находился в кармане.

Из каждого замера был выбран интервал длиной 256 отсчётов, что примерно соответствует 2,5 секундам реального времени.

### 1.2. Параметризация

Для параметризации телеметрических данных использовался метод, предложенный в работе [11]. Для выбранных на предыдущем шаге интервалов вычислялись детализирующие и аппроксимационные коэффициенты первого уровня. На основе этих коэффициентов рассчитывались следующие статистические параметры: среднее значение, минимум, максимум и стандартное отклонение. Составленный из них вектор использовался для обучения сети. Схема параметризации представлена на рисунке 1.

### 1.3. Выбор типа нейросети и её параметров

Для решения задачи классификации была выбрана двухслойная сеть прямого распространения, так как такие сети обладают высокой способностью к обобщению [12] и успешно используются для распознавания образов. В качестве функции активации нейронов использовалась функция tanh, поскольку она обеспечивает более быстрое обучение, чем другие функции [13].

Количество нейронов во входном и выходном слоях совпадает с размерностями входа и выхода нейросети и, соответственно, равно 8 и 2. Количество нейронов в скрытом слое, равное 4, было подобрано экспериментально в соответствии с рекомендациями из [13].

Для обучения «с учителем» использовалась модификация алгоритма обратного распространения ошибки - алгоритм RProp (Resilient Propagation) [14]. RProp является одним из самых быстрых алгоритмов обучения сетей прямого распространения [15].

### 1.4. Обучение нейросети

Всё множество параметризованных данных было разделено на три множества: обучающее, тестовое и множество «нарушителей» (множество, которое

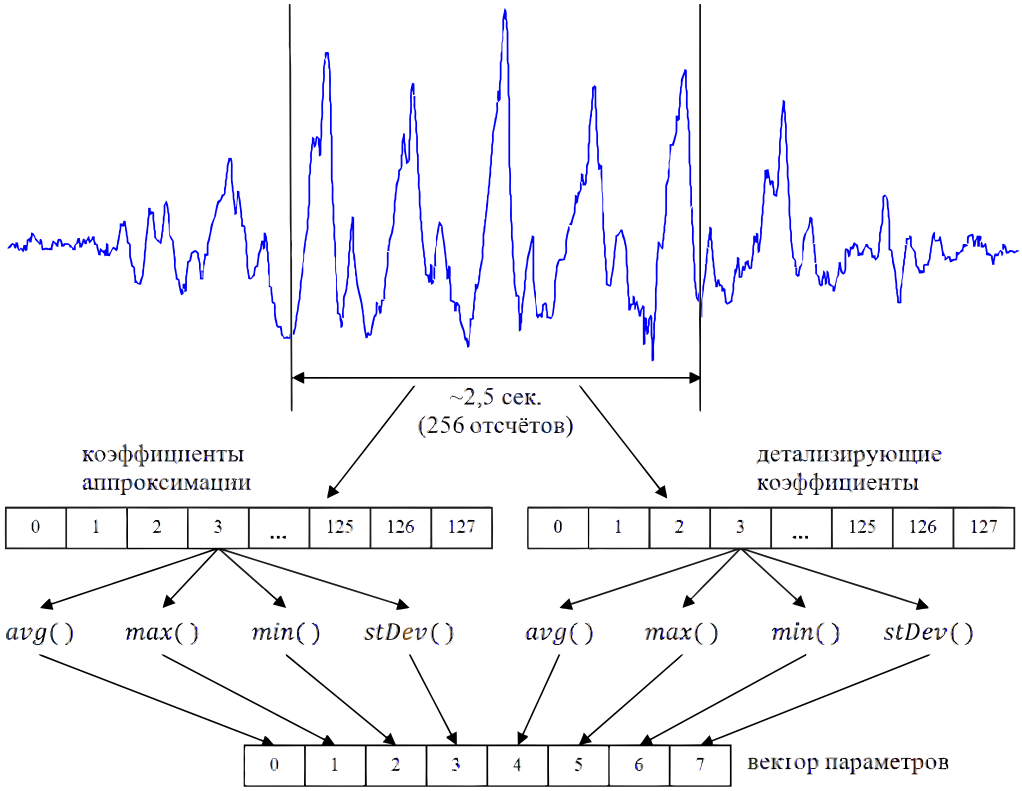


Рис. 1. Схема параметризации

сеть не должна распознавать). 60 измерений первой группы разделили пополам между обучающим и тестовым множествами. В множество «нарушителей» вошли измерения из второй группы.

Для нормирования входных значений нейросети использовалась формула

$$f(x) = \frac{x - min}{max - min}(high - low) + low,$$

где $x$ - значение, подлежащее нормированию; $min$ - минимально возможное значение $x$; $max$ - максимально возможное значение $x$; $low = -1$ - нижний предел интервала нормированных значений; $high = 1$ - верхний предел [16, с. 149].

Эталонные выходные значения нейросети кодировались таким образом, чтобы евклидовы расстояния между их кодами были равны. Для группы из трёх человек кодовые расстояния между эталонами составили 1,73 (при использовании нормирования в интервале от -1 до 1). В результате этого ошибочная идентификация, к примеру, участника 1 как участника 2, будет иметь такой же вес, что и ошибочная идентификация 1 как 3 [16, с. 152].

Ошибка обучения вычислялась как кодовое расстояние между реальным выходом сети и эталонным:

$$d = \sqrt{(i_1 - a_1)^2 + (i_2 - a_2)^2 + ... + (i_n - a_n)^2},$$

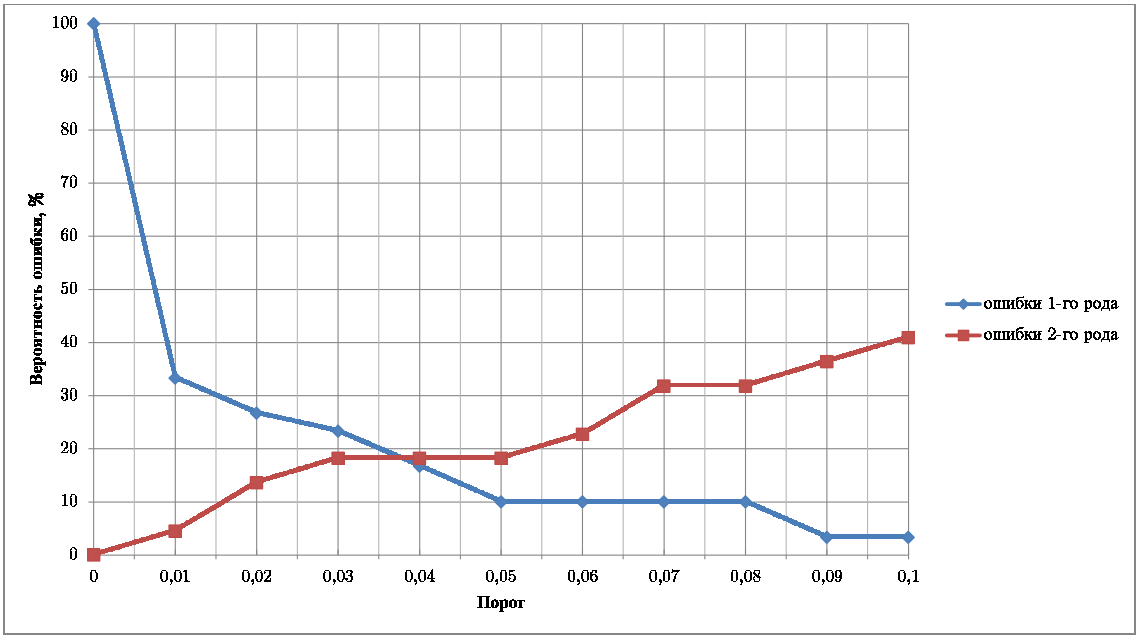


Рис. 2. График зависимости вероятностей ошибок от порогового значения

где $a$ - реальный выход, а $i$ - эталонный [16, с. 153]. В целях предотвращения переобучения сети ошибка рассчитывалась не только на данных из обучающего множества, но и на одном наборе данных из тестового множества [17].

Для параметризации, экспериментальным способом, был выбран вейвлет Добеши D6.

Обучение проводилось до тех пор, пока ошибки идентификации 1-го, 2-го и 3-го участников не становились меньше 0,01.

В качестве порогового значения ошибки было выбрано значение 0,05. Если евклидово расстояние между выходом нейросети и эталоном меньше этого порогового значения, то считается, что образец был идентифицирован. График зависимости вероятностей ошибок от порогового значения изображён на рисунке 2.

Таблица 1. Результаты тестирования обученной нейросети

| Вероятность корректного распознавания участника 1, % | Вероятность корректного распознавания участника 2, % | Вероятность корректного распознавания участника 3, % | Вероятность ошибки 2-го рода, % |
|---|---|---|---|
| 100 | 90 | 80 | 18,2 |
| 90 | | | |

### 1.5. Результаты

В таблице 1 представлены результаты тестирования нейросети, обученной в соответствии с разработанным методом.

Отметим, что полученный результат выполнен на относительно коротких выборках, которые включали 4 – 5 полных шага. С учётом того, что первый шаг и последний искажены началом и остановкой движения, для идентификации использовались 2 – 3 полных шага. Следует ожидать улучшения идентификации при сборе более длинных записей.

Выбранный метод параметризации может быть модифицирован, что тоже может привести к улучшению точности идентификации.

## Заключение

В данной дипломной работе был разработан и реализован метод идентификации личности по походке с использованием вейвлет-параметризации и нейронной сетью в качестве классификатора.

Для этого было сделано следующее:

- разработана библиотека вейвлетов JWT, исходный код библиотеки размещён в SVN-хранилище Омской группы пользователей Java по адресу: `http://java.net/projects/omskjug/sources/svn/show/trunk/JWT`;
- разработан метод параметризации на основе дискретного вейвлет- преобразования;
- эмпирически подобран вейвлет (Добеши D6), дающий лучшее распознавание;
- выбран тип нейросети (сеть прямого распространения), алгоритм её обучения (RProp) и подобраны параметры нейросети и алгоритма обучения;
- проведено обучение нейросети.

Вероятность корректной идентификации по походке составила в среднем 90%, а вероятность ошибки 2-го рода — 18,2%.

## Литература

1. Биометрия // Википедия: свободная электронная энциклопедия: на русском языке. URL: `http://ru.wikipedia.org/wiki/Биометрия` (дата обращения: 10.10.2010).
2. Ekinci M. Human Identification Using Gait // Turk J Elec Engin. 2006. V.14, N.2. P. 267–291.
3. Mantyjarvi J. et al. Identifying Users of Portable Devices from Gait Pattern with Accelerometers // 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing. V. 2. 2005. P. 973–976.
4. Sprager S., D. Zazula. A Cumulant-Based Method for Gait Identification Using Accelerometer Data with Principal Component Analysis and Support Vector Machine // WSEAS Transactions on Signal Processing. 2009. V.5, N.11. P. 369–378. 37

5. Java // Википедия: свободная электронная энциклопедия: на русском языке. URL: http://ru.wikipedia.org/wiki/Java (дата обращения: 10.10.2010).
6. Encog Java and DotNet Neural Network Framework. URL: http://www.heatonresearch.com/encog (дата обращения: 12.11.2010).
7. Benchmarking and Comparing Encog, Neuroph and JOONE Neural Networks. URL: http://www.codeproject.com/KB/recipes/benchmark-neuroph-encog.aspx (дата обращения: 12.11.2010).
8. Sun SPOT // Wikipedia: The Free Encyclopedia. URL: http://en.wikipedia.org/wiki/Sun_SPOT (дата обращения: 12.11.2010).
9. IEEE 802.15.4-2006 // Википедия: свободная электронная энциклопедия: на русском языке. URL: http://ru.wikipedia.org/wiki/IEEE_802.15.4-2006 (дата обращения: 20.11.2010).
10. Goldman R. Using the LIS3L02AQ Accelerometer. URL: http://www.sunspotworld.com/docs/AppNotes/AccelerometerAppNote.pdf (дата обращения: 22.12.2010).
11. Tayel M.B., El-Bouridy M.E. ECG Images Classification Using Feature Extraction Based on Wavelet Transformation and Neural Networks // ICGST, International Conference on AIML. 2006.
12. Вежневец А. Популярные нейросетевые архитектуры // Компьютерная графика и мультимедиа. 2004. N.2. URL: http://cgm.computergraphics.ru/content/view/57 (дата обращения: 21.11.2010).
13. Swingler K. Applying Neural Networks. A Practical Guide. URL: http://www.nsu.ru/matlab/MatLab_RU/neuralnetwork/book4/3_2.asp.htm#3_2 (дата обращения: 23.11.2010).
14. Локальные минимумы. Выбор длины шага обучения. Динамическое добавление нейронов. Алгоритм RProp. Обучение без учителя. Алгоритм Хебба. URL: http://apsheronsk.bozo.ru/Neural/Lec4.htm (дата обращения: 23.11.2010).
15. RProp // Wikipedia: The Free Encyclopedia. URL: http://en.wikipedia.org/wiki/Rprop (дата обращения: 23.11.2010).
16. Heaton J. Programming Neural Networks with Encog 2 in Java. St. Louis : Heaton Research, 2010.
17. Чубукова И.А. Data Mining. URL: http://www.intuit.ru/department/database/datamining/11/4.html (дата обращения: 10.10.2010).

*Математические структуры и моделирование*
2011, вып. 23, с. 38–42

УДК 004.93'1

# ИСПОЛЬЗОВАНИЕ НОРМИРОВАННЫХ КАДРОВ СИГНАЛА В ЗАДАЧЕ РАСПОЗНАВАНИЯ ПО ПОХОДКЕ

**Е.А. Первушин, Д.Н. Лавров**

В статье рассматривается задача распознавания человека по показаниям телеметрического датчика во время ходьбы. Описывается метод извлечения признаков, рассматривающий сигнал как периодическую функцию и выделяющий участки, соответствующие периодам. Строится система идентификации на основе метода ближайшего соседа, и приводятся результаты экспериментов.

## Введение

Предположим, некоторое устройство (например, мобильный телефон) содержит акселерометр, то есть способно получать данные об ускорении. Применяя устройство в качестве датчика, фиксирующего телеметрические показания, такие данные могут быть использованы для распознавания человека по его действиям, например, по походке.

Примеры записи датчика при ходьбе приведены на рисунке 1. Здесь показания датчика дискретизованы по времени с частотой дискретизации 100 Гц. По оси Y откладываются значения модуля ускорения. В приведённых примерах датчик был фиксирован в одном из трёх положений: на руке возле кисти, на бедре (в кармане брюк), на ноге возле ступни. Квазипериодическая структура графика соответствует повторению шагов. Поэтому было решено применить к такому сигналу метод, описанный в [1] и применённый для обработки речевого сигнала.

## 1. Обработка сигнала

В данном методе сигнал рассматривается как последовательность независимых элементов, каждый из которых представляет собой результат повторяющегося во времени события (или одного из некоторого множества событий). Тогда, определив алгоритм выделения таких элементов и алгоритм вычисления меры сходства/различия между ними, становится возможным сравнивать сигналы произвольной длительности.


Омский государственный университет им. Ф.М. Достоевского
E-mail: pervushin_evgen@mail.ru, lavrov@omsu.ru

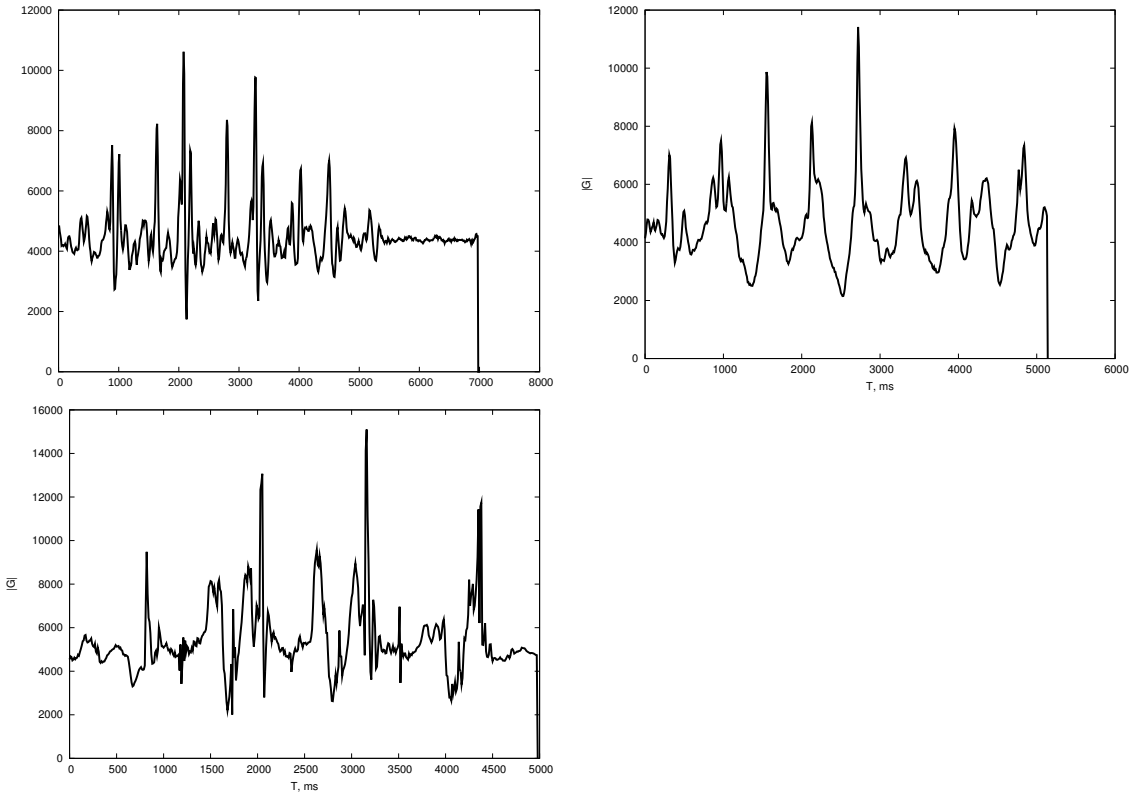


Рис. 1. Пример показаний датчика при ходьбе. Положения датчика сверху вниз: на бедре, на руке, на ноге

Для выделения участков, соответствующих периодам (шагам), использовался алгоритм, основанный на вычислении функции среднего значения разности [2]. Данный алгоритм требует задания диапазона возможных значений периодов. Минимальное значение периода было принято равным 450 мс, максимальное — 800 мс. Алгоритм последовательно исследует участки сигнала, попадающие в окно размера $2 \cdot 800$ мс. На каждом участке в случае принятия решения о наличии периода определяется длина периода $L$ (выраженная в количестве отсчётов) и находится точка максимума, которая затем используется для определения начала кадра. Для каждого периода фиксируется начало так, чтобы извлечённые кадры могли быть сравнены с помощью обычной меры расстояния, например, с помощью евклидова расстояния. Выбор начала кадра определялся фиксированным по времени относительно точки максимума так, чтобы выделить наиболее устойчивые элементы. Более высокие результаты были достигнуты при значении сдвига, равном 200 мс слева от точки максимума. Относительно выбранного начала следующие $L$ отсчётов представляют собой извлекаемый кадр, который затем нормируется по амплитуде для компенсации вариабельности между кадрами во времени. Нормализация кадров осуществлялась следующим образом:

$$x'_i = \frac{1}{A}(x_i - c), i = 1, \ldots, L,$$

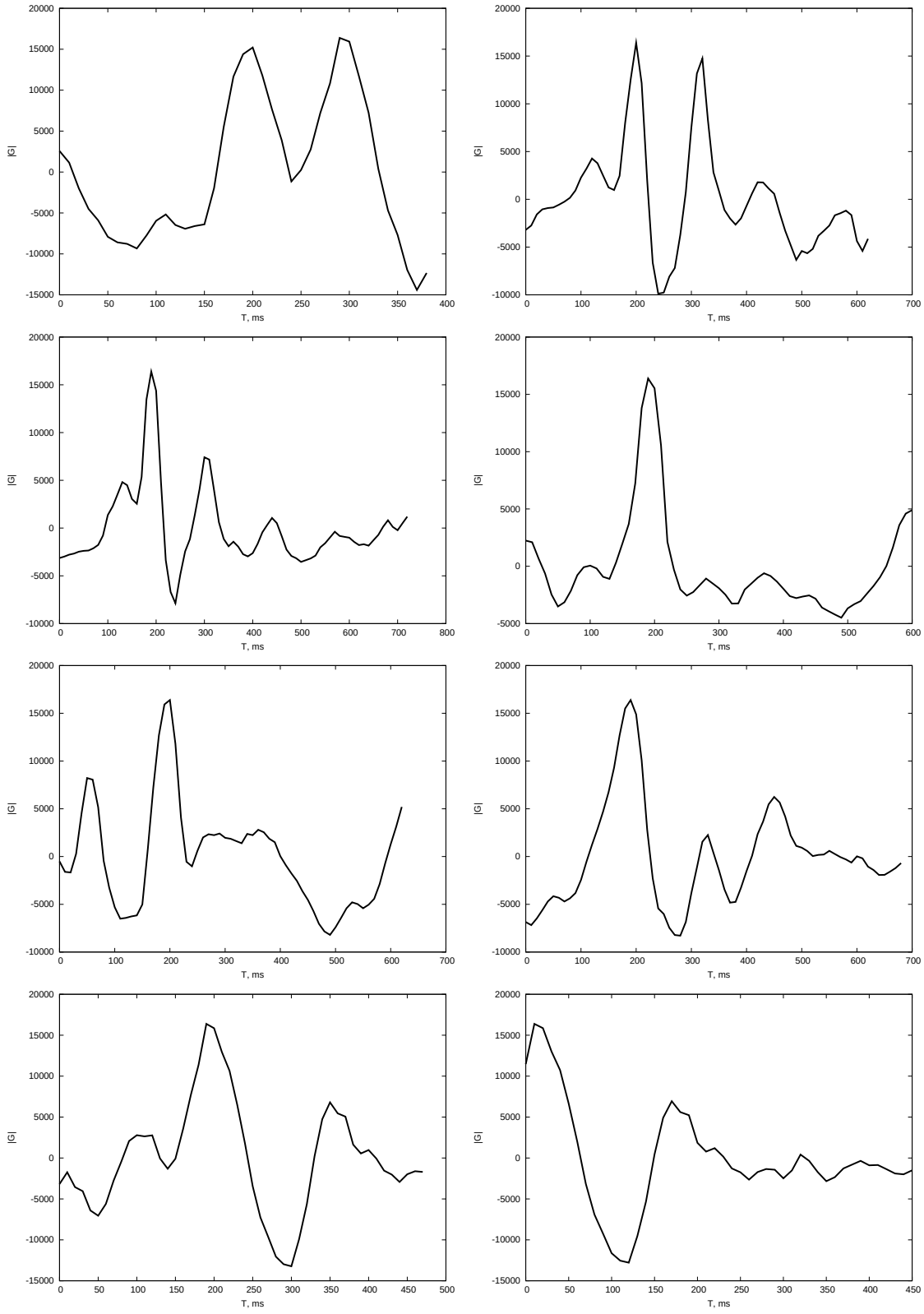


Рис. 2. Пример извлечённых кадров. Положение датчика: на бедре

$$c = \frac{1}{L}\sum_{i=1}^{L} x_i,$$

$$A = \max_i |x_i - c|,$$

где $x'_1, \ldots, x'_L$ — отсчёты нормированного кадра, $x_1, \ldots, x_L$ — оригинальные отсчёты кадра. Пример извлечённых кадров можно видеть на рисунке 2.

## 2. Система идентификации

Для идентификации был использован метод ближайшего соседа. В этом методе модель представляется самими данными, а для классификации вычисляются кратчайшие расстояния до сохранных моделей. Расстояние между двумя кадрами вычислялось по формуле

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\frac{1}{L}\sum_{i=1}^{L}(x_i - y_i)^2}$$

$$L = \min(L_1, L_2),$$

где $L_1$, $L_2$ — количество отсчётов в кадрах $\mathbf{x}$ и $\mathbf{y}$. Необходимо отметить, что для использования такого расстояния частота дискретизации сравниваемых кадров должна совпадать. Кратчайшие расстояния от каждого кадра идентифицируемой последовательности затем суммируются. Пусть $\mathbf{x}_1$,...,$\mathbf{x}_L$ — кадры, извлечённые из представленного для идентификации образца. Согласно методу ближайшего соседа правило для классификации данной последовательности можно записать в следующем виде:

$$C = \arg\min_{C_k} \sum_{i=1}^{L} \min_{\mathbf{x}_j \in C_k} d(\mathbf{x}_i, \mathbf{x}_j),$$

где $C_k$ — модели зарегистрированных пользователей.

## 3. Эксперименты и результаты

Описанная система была испытана с помощью эксперимента по идентификации на замкнутом множестве. Эксперименты проводились как отдельно для каждого положения датчика, так и для совместного использования показаний. В последнем эксперименте для построения моделей пользователей использовались данные записей с разными положениями датчика. Продолжительность записей, используемых для создания шаблонов и для попыток идентификации, составляет 3 – 5 шагов. Записи каждого пользователя были сделаны в разные сессии, интервал между которыми составляет от 3 до 10 дней. Записи одной сессии использовались для создания моделей, записи остальных — для попыток идентификации. Данный эксперимент был использован для настройки параметров, используемых в вышеописанных алгоритмах. Результаты достигнутой точности приведены в таблице 1.

Таблица 1. Точность идентификации по походке

| **Расположение датчика** | **Количество пользователей** | **Количество верных идентификаций** |
|---|---|---|
| На бедре | 5 | 5/6 (83,3%) |
| На руке | 6 | 11/14 (78,6%) |
| На ноге | 6 | 5/10 (50,0%) |
| Использование всех данных | 7 | 21/30 (70,0%) |

## 4. Заключение

Предложенный метод идентификации оказался применим к задаче распознавания личности по показаниям телеметрического датчика во время ходьбы. Для более точной оценки качества работы метода необходимо расширить базу показаний акселерометров. Наибольшую точность метод демонстрирует при работе с показаниями снятыми при положении датчика на бедре.

## Литература

1. Первушин Е.А., Лавров Д.Н. Система идентификации диктора на основе выделения информативных участков речевого сигнала // Информационные технологии и автоматизация управления. Матер.II межвуз. науч.-практ. конф. ОмГТУ. Омск : Изд-во ОмГТУ. 2010. С. 188-189.
2. Первушин Е.А., Лавров Д.Н. Алгоритм выделения основного тона и детектирования тон/не тон по минимумам разностной функции на участке минимального периода // Математические структуры и моделирование. Омск : Омский государственный университет им. Ф.М. Достоевского. 2010. Вып.22. С. 24-27.

*Математические структуры и моделирование*
2011, вып. 23, с. 43–48

УДК 004.934

# АВТОМАТИЧЕСКАЯ СЕГМЕНТАЦИЯ РЕЧЕВОГО СИГНАЛА НА БАЗЕ ДИСКРЕТНОГО ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЯ

**О.А. Вишнякова, Д.Н. Лавров**

В данной статье предложен метод сегментации речевого сигнала, основанный на анализе вариации уровня энергии вейвлет-спектра. Расстановка границ происходит на участках быстрого изменения огибающей энергии сигнала сводно по всем уровням детализации.

## Введение

Одной из важнейших задач в системах автоматической обработки речи является задача сегментации в соответствии с фонетической транскрипцией языка. Для голосовой верификации характерные признаки голоса должны вычисляться на определённых сегментах речевого сигнала. Так, частота основного тона, присущая диктору, должна вычисляться на гласноподобных участках сигнала, форма речевого тракта характеризуется формантными частотами, измеряемыми на известных гласных звуках, скорость артикуляции определяется по длительностям переходных процессов между артикуляторно-акустическими сегментами. [1] Сегментация необходима при решении обратной задачи — восстановления формы речевого тракта по акустическому сигналу [2], которая может быть использована в следующих областях: системы сжатия и передачи речи в мобильной телефонии [3], синтезаторы речи по произвольному тексту [4], системы автоматического распознавания речи, системы обучения иноязычному произношению.

В исследовательских системах и на этапе предварительной разработки возможно использование ручной сегментации. Однако она требует значительных затрат сил и времени: во-первых, в слитной речи нет пауз между словами, во-вторых, коартикуляция, возникающая и на границе последовательно производимых звуков, которая существенно облегчает правильное восприятие и понимание речи, но затрудняет задачу поиска границ сегментов. Кроме того, практически невозможно точно воспроизвести результаты ручной сегментации вследствие субъективности человеческого слухового и зрительного восприятия.


Омский государственный университет им. Ф.М. Достоевского
E-mail: olga@infotekorg.ru, lavrov@omsu.ru

Подобных проблем не возникает при автоматической сегментации, которая также небезошибочна, но даёт воспроизводимые результаты.

Существует два основных типа алгоритмов сегментации речи. К первому типу относятся алгоритмы, которые производят сегментацию речи при условии, что известна последовательность фонем данной фразы. Другой тип алгоритмов не использует априорной информации о фразе, и при этом границы сегментов определяются по степени изменения акустических характеристик сигнала. При автоматической сегментации желательно использовать только общие характеристики речевого сигнала, поскольку обычно на этом этапе нет конкретной информации о содержании речевого высказывания

## 1. Сегментация с использованием кратномасштабного анализа

Как известно, речевой сигнал состоит из квазистационарных участков, соответствующих голосовым и шипящим фонемам, перемежаемых участками со сравнительно быстрыми изменениями спектральных характеристик сигнала (межфонемные переходы, взрывные и смычные фонемы, внутрисловные переходы речь-пауза) [5]. В пределах стационарных участков значительную роль для анализа речевого сигнала играют спектральные особенности сигнала, определяемые передаточной характеристикой речевого тракта, изменяющейся в процессе артикуляции. Можно сказать, что речевой сигнал характеризуется нелинейными флуктуациями различных масштабов. Поэтому весьма эффективным для анализа речевого сигнала представляется кратномасштабный анализ и вейвлет–преобразование.

Вейвлет-разложение речевого сигнала длины N отсчётов представляет собой сумму:

$$f(t) = \sum_{k=0}^{N/2^n-1} s_{nk}\varphi_{nk} + \sum_{j=1}^{N}\sum_{k=0}^{N/2^n-1} d_{jk}\psi_{jk},$$

$$\varphi_{nk} = 2^{j/2}\varphi(2^j t - k), \text{ где } j,k \in \mathbb{Z}$$

$$\psi_{jk} = 2^{j/2}\psi(2^j t - k), \text{ где } j,k \in \mathbb{Z},$$

где $n$ – количество уровней декомпозиции, $s_{nk}$,$d_{jk}$ – коэффициенты аппроксимации и детализации вейвлет-разложения, $\varphi$ – скейлинг (масштабная) функция, $\psi$ – базисный («материнский») вейвлет.

Так как вейвлет-коэффициенты аппроксимации соответствуют передаточной характеристике фильтра низких частот, а детализации — высокочастотному фильтру, то можем рассматривать поведение речевого сигнала в различных частотных диапазонах.

Частотный диапазон ниже 125Гц не используется, т.к. не содержит информации, важной для задачи сегментации. Это обусловлено природой человеческой речи, охватывающей интервал 150 – 4000 Гц. Таким образом, достаточно 6 уровней разложения.

Таблица 1. Частотные диапазоны

| Уровень детализации | Частотный диапазон Добеши16 | Частотный диапазон Мейера |
|---|---|---|
| уровень 1 | 2000Гц - 4000Гц | 2756Гц - 5512Гц |
| уровень 2 | 1000Гц - 2000Гц | 1378Гц - 2756Гц |
| уровень 3 | 500Гц - 1000Гц | 689Гц - 1378Гц |
| уровень 4 | 250Гц - 500Гц | 345Гц - 689Гц |
| уровень 5 | 125Гц - 250Гц | 172Гц - 345Гц |
| уровень 6 | | 86Гц - 172Гц |

## 2. Алгоритм сегментации

Сегментация речевого сигнала подразумевает выделение участков сигнала, соответствующим отдельным структурным единицам. Если в качестве таких единиц рассматривать фонемы, то задача сегментации сводится к обнаружению межфонемных переходов. В рамках традиционных подходов решение этой задачи весьма проблематично. Однако вейвлет-преобразование (DWT) позволяет решить эту проблему, по крайней мере, для фонем, соответствующих сравнительно протяженным квазистационарным участкам. Дело в том, что на межфонемных переходах сигнал претерпевает значительные изменения сразу на многих масштабах исследования и, соответственно, характеризуется возрастанием вейвлет-коэффициентов для многих уровней детализации, в то время как на стационарных участках фонем вейвлет-коэффициенты оказываются сгруппированными вблизи определённых масштабов [7]. Таким образом, отыскание межфонемных границ может быть сведено к отысканию моментов увеличения вейвлет-коэффициентов на значительном количестве уровней масштабирования. При этом существенным является выбор вейвлетного базиса, который должен позволять описывать стационарный речевой сигнал со сравнительно малым числом ненулевых коэффициентов. Возможно использование нескольких вейвлетных базисов для поиска межфонемных переходов в каждом из них с последующим объединением результатов [6].

Для начала сигнал разбивается на перекрывающиеся участки, к каждому из которых применяется DWT. Для каждого фрейма $i$ и уровня декомпозиции $n$ можно определить энергию:

$$E_n(i) = \sum_{j=1}^{2^n-1} d^2_{n,j+2^{n-1}i}, \text{ где } i = 0, ..., 2^{-M}N - 1. \quad (1)$$

Энергия сигнала (1) быстро меняется от фрейма к фрейму для каждого уровня из-за неизбежных шумов во время записи речевого сигнала. Для сглаживания определяем $E'_n$, заменяя значение $E_n$ в окне шириной 3 – 5 фреймов на максимальное значение $E_{max}$ в этом окне. Для определения скорости изменения энергии вычисляем производную $R$. Межфонемные переходы характеризуются небольшими, но быстрыми изменениями уровня энергии на одном или бо-

лее уровнях детализации. Таким образом, критерием выбора границы фонемы должно быть быстрое изменение производной при невысоком уровне энергии. Иными словами, для каждого уровня детализации мы ищем такие участки, на которых значение производной близко по своим абсолютным значения к уровню энергии на интервале, при этом разница не превышает некоторого порогового значения $div_{opt}$, а энергия на этом интервале обязательно более чем $E_{min}$ как гарантия анализа именно речевого сигнала, а не шумового участка:

$$div_{opt} \geqslant ||R_n(i)| - E'_n(i)|$$

## 3. Доработки алгоритма

Требуются некоторые доработки алгоритма для более точного определения границ сегментов. Положение границ может различаться между уровнями. Это объяснимо природой вейвлет-преобразования — рассмотрение сигнала на различных частотных диапазонах. Так для части фонем только один из уровней покажет значительное изменение энергии, для остальных — несколько. Таким образом, на каждом уровне определяется только часть межфонемных переходов и необходима группировка результата. При этом межфонемный интервал не может быть менее порогового значения — минимальной длительности фонемы. Порог установлен в 25 мсек. Общий алгоритм сегментации:

1. В качестве предобработки сигнал нормализуется: все отсчёты делятся на максимальное значение, для установки единых пороговых значений для любых входных сигналов.

2. Входной сигнал разбивается на фреймы по 256 отсчётов при частоте дискретизации 16 кГц с перекрытием от 25% до 50%.

3. Каждый фрейм накрывается окном Хэмминга для устранения дефектов на краях.

4. К каждому фрейму применяется вейвлет-преобразование. Используется разложение до 6-го уровня декомпозиции.

5. Для каждого уровня декомпозиции определяется энергия, как сумма квадратов значений коэффициентов детализации $E$ (1).

6. Так как энергия сильно меняется от фрейма к фрейму из-за неизбежного шума, необходимо сглаживание. Для этого вычисляется усреднённая энергия $E'_n$ для каждого уровня декомпозиции, заменяя значение энергии на максимальное $E_{max}$ для каждых 3 на первых трёх, и на каждых 5 для последующих уровней детализации.

7. Для определения скорости изменения энергии вычисляется производная $R$.

8. Критерии выбора границ фонем:

$$div_{opt} > ||R_n(i)| - E'_n(i)|$$

$$div_{opt} < ||R_n(i+1)| - E'_n(i+1)| \text{или } div_{opt} < ||R_n(i-1)| - E'_n(i-1)|$$

$$E'_n(i) > E_{min}$$

9. Для объединения результатов расстановки границ между уровнями все индексы объединяются в один вектор. Чтобы избежать ложных границ, устанавливается минимальный интервал фонемы — 28 мсек.

## 4. Результаты экспериментов

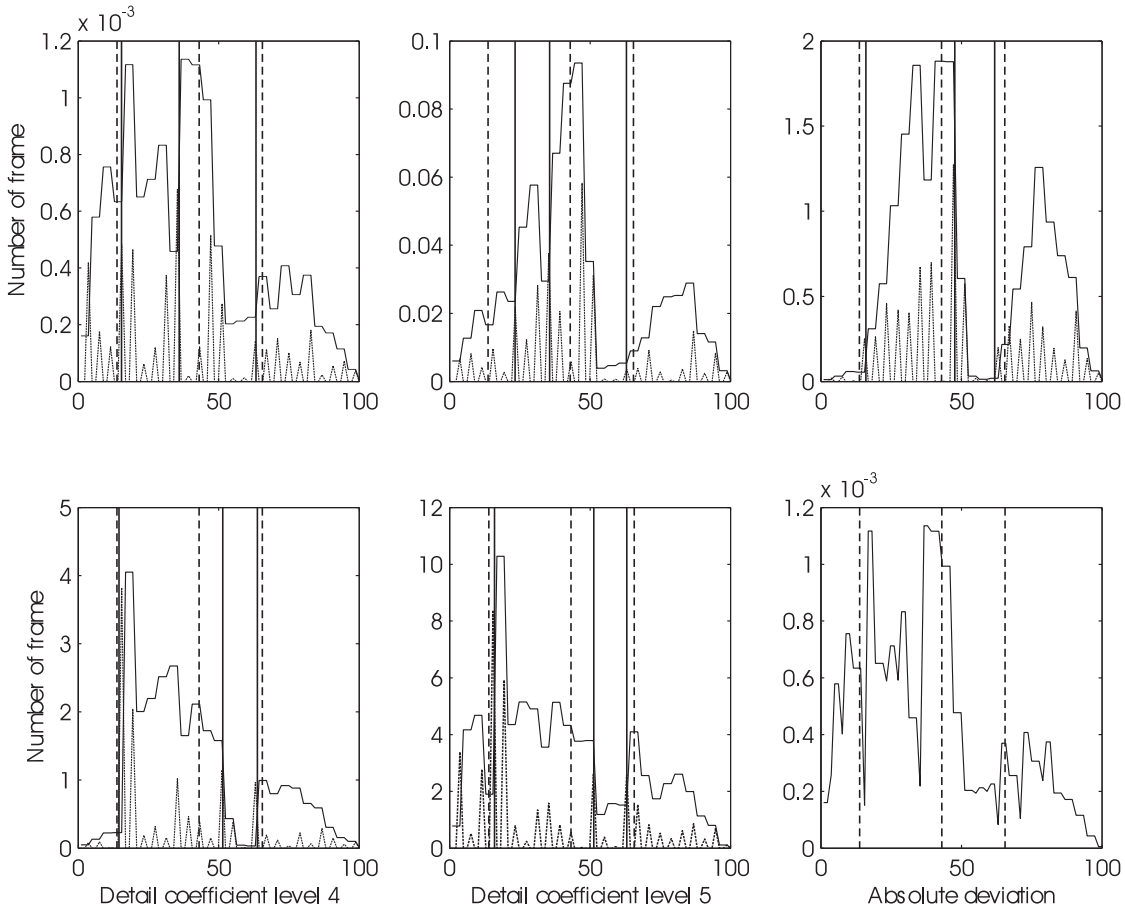


Рис. 1. Пример сегментации слова «мама»

Для экспериментов использовано 35 различных дифонов и трифонов, записанных при частоте дискретизации 16кГц.

На рисунке 1 пунктирными линиями отмечены границы ручной разметки, сплошными — автоматической. Приведены графики энергий и их производных для каждого уровня детализации.

Экспериментально оптимальными пороговыми значения выбраны $div_{opt} = 0.03$, $E_{min} = 0.005$. При этом с увеличением порогового коэффициента уменьшается чувствительность алгоритма к изменениям речевого сигнала. Так, при значениях 0.01 – 0.02 заметно выделение лишних сегментов для гласных, хорошо разделяются голосовые звуки, стоящие рядом «oa», «au». При больших значениях порога количество лишних сегментов мало, но перестают разделяться голосовые звуки. Результаты экспериментов показали незначительную разницу в эффективности вейвлетов Майера, Добеши 16, Добеши 8, Симлета 6 порядка. Это говорит о возможности применения всех их в качестве базиса разложения с возможным будущим объединением результата для повышения уровня распознавания границ сегментов.

## 5. Заключение

Предложенный метод сегментации основан на дискретном вейвлет-преобразовании. Эффективность метода обусловлена природой речевого сигнала — изменения уровня энергии для ряда фонем проявляются только в узком диапазоне частот. Именно поэтому границы вероятнее детектировать, анализируя значения энергий поддиапазонов вейвлет-разложения, а не сигнала в целом как в случае преобразований, основанных на Фурье анализе. В качестве основного параметра определения точной границы сегмента используется скорость изменения энергии при последующем объединении результатов расстановки границ между уровнями детализации.

### Литература


1. Рамишвили Г. С. Автоматическое опознавание говорящего по голосу. М. : Радио и связь, 1981. 224 с.
2. Макаров К.С. Построение и исследование артикуляторных кодовых книг для решения речевых обратных задач : Диссер. на соиск. степ. к.т.н., ИППИ РАН, 2005.
3. Leonov A.S., Sorokin V. N. Inverse problem for the vocal tract: identification of control forces from articulatory movements // Pattern Recognition and Image Analysis. 2000. V.10, №1. P.110-126.
4. Сорокин В.К. Синтез речи. М. : Наука, 1992. 392 с.
5. Сорокин В.Н., Цыплихин А.И. Сегментация и распознавание гласных // Информационные процессы. 2004. т.4, № 2. с. 202-220.
6. Ziolko B., Manandhar S., Wilson R., Ziolko M. Wavelet method of speech segmentation // Proceedings of 14th European Signal Processing Conference EUSIPCO. 2006.
7. Ермоленко Т., Шевчук В. Алгоритмы сегментации с применением быстрого вейвлет-преобразования // Статьи, принятые к публикации на сайте международной конференции Диалог'2003. URL: `http://www.dialog-21.ru`.

*Математические структуры и моделирование*
2011, вып. 23, с. 49–57

УДК 004.414.2:004.04

# ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ ПЕРИОДИЧЕСКИХ ИЗДАНИЙ

**И.П. Бесценный, Е.Н. Султанкин**

Обосновывается методология объектно-ориентированного анализа (ООА) и ее применение к информационной системе для периодических изданий.

## Введение

Объектно-ориентированная технология широко применяется при разработке программного обеспечения больших информационных систем, но она до сих пор находится на стадии становления. Особенно это касается первой стадии процесса разработки – анализа предметной области. Большие, сложные системы в принципе не поддаются формальному описанию. При анализе предметная область разделяется на объекты, обладающие определёнными свойствами и вступающие во взаимодействие между собой. Поиск «правильных» категорий объектов требует больших усилий, при этом само понятие «правильности» не удаётся чётко сформулировать. Очень непросто выработать устойчивую систему абстракций, определяющую соответствующие объекты, описать их взаимодействие. При этом надо предвидеть сложности реализации объектов, обеспечивающей их повторное использование при проектировании других систем. Именно повторное использование и простота модификации гарантирует эффективность объектно-ориентированного программирования.

Многие исследователи разрабатывали каждый свою методику объектно-ориентированного анализа предметной области. Примеры этих методик описаны в [1, 2, 4, 5, 9]. Большая часть литературы англоязычна, и лишь немногие книги переведены на русский язык. Возможно, это объясняется малой популярностью в России теоретических изысканий по сравнению с практическими рекомендациями. Как правило, в учебниках по объектно-ориентированному программированию предлагаются законченные проекты и не говорится о том, как они были получены. Кроме того, современные средства автоматизации стадий анализа и проектирования информационных систем очень дорогие для российского пользователя, и возможности познакомиться с принципами их работы весьма ограничены. Всё это приводит к недостаточной образованности


Омский государственный университет им. Ф.М. Достоевского
E-mail: ibests@mail.ru, sultankin_evgeni@mail.ru

российских студентов в области методов объектно-ориентированного анализа и объектно-ориентированного проектирования. Сейчас на факультете компьютерных наук одним из авторов разрабатывается новый спецкурс, посвящённый анализу и проектированию информационных систем, и решается проблема приобретения программного обеспечения для CASE-технологий.

В настоящей статье дан обзор и показано применение принципов ООА как первого этапа проектирования автоматизированной информационной системы, предназначенной для координации издательской деятельности, на примере вымышленной редакции газеты.

## 1. Современные методики ООА

Различные методики ООА имеют общие черты, но отличаются акцентами на определённые аспекты процесса моделирования предметной области. Общая терминология обуславливает взаимопроникновение идей и понятий всех методов. В результате объединения этих методик были созданы стандарт описания моделей IDEF [3] и язык моделирования UML [5, 6].

Основная процедура ООА – построение информационной модели, в которой фундаментальная структура предметной области абстрагируется от реального мира и чётко формализуется и документируется. Большинство исследователей выделяют три разновидности фундаментальной структуры предметной области. Функциональная структура – упорядоченная совокупность действий, процессов и операций, производимых анализируемой системой. Компонентная структура состоит из элементов (объектов) и их взаимосвязей. Динамическая (поведенческая) структура описывает возможные состояния системы и события, вызывающие переход из одного состояния в другое. Они могут анализироваться все вместе и должны быть согласованы друг с другом.

Для отдельных областей применения ООА приобретают значение распределение объектов и процессов во времени и пространстве (например, при разработке систем реального времени) или структура ограничений и правил поведения и использования (например, для бизнес-процессов). Конечная цель ООА – представить предметную область в терминах классов и объектов для дальнейшего проектирования логической структуры разрабатываемого программного обеспечения. Поэтому при анализе планируется наряду с вышеперечисленными концептуальная структура, объединяющая функции, компоненты, события, сценарии поведения с абстрактными понятиями, характерными для понимания сущности предметной области в иерархию классов.

Средствами описания функциональной структуры являются контекстные диаграммы, диаграммы процессов и потоков (DFD), снабжённые спецификацией используемых имён. Основная методика анализа функциональной структуры приведена в [7, 8]. Отметим необходимость иерархической организации диаграмм по вложению для удобства представления большого количества объектов и связанную с этим проверку логической непротиворечивости составных частей. Эти трудоёмкие процедуры лучше производить с использованием специальных компьютерных программ моделирования. Работа с такими программами

описана в [10]. Мы использовали демонстрационную версию CASE Studio.

Параллельно с функциональной структурой разрабатывается структура используемых данных. Известные диаграммы «сущностей-связей» (ERD) помогают отразить связи, невидимые на DFD, и дополняют их. В дальнейшем они используются для реализации в СУБД. Для отображения динамического поведения системы используются диаграммы переходов состояний (STD), диаграммы деятельности языка UML, которые позволяют выделить основные механизмы реагирования на события. Дальнейшее структурирование множества операций снова приводит к DFD, но уже более детализированным. Таким образом процесс анализа становится итеративным. После проверки полноты и непротиворечивости модели создаются предварительные диаграммы классов и объектов, служащие отправной точкой для следующего этапа разработки – объектно- ориентированного проектирования.

## 2. Описание моделируемого издательства

Вымышленная редакция газеты нуждается в оперативном реагировании на происходящие события, чтобы вовремя создавать репортажи с места событий и составлять на основе этих репортажей выпуски газеты. Для этого она контактирует с различными агентами: информаторами и корреспондентами, а также поддерживает обратную связь с читателями посредством традиционной почты и различных сервисов сети Internet.

Газета выходит в различных вариантах:

- «Утренний Отчёт» – утренняя газета, доставляемая распространителями и содержащая краткие обзоры дневных выпусков и анонсы событий;
- «Рабочий Полдень» – ежедневная газета небольшого формата, которую можно купить в киосках или получить по подписке;
- «Вестник Интернационала» – еженедельный выпуск расширенного формата, заменяющий собой пятничный номер;
- «Интернационал советует...» – отдельные приложения, выпускающиеся раз в месяц на актуальную тему;
- Интернет–версия – здесь публикуется архив новостей за всё время существования газеты (полная версия номера появляется через неделю после выхода в печать).

Источниками информации для этих выпусков являются сообщения о событиях разной степени достоверности: от непроверенных слухов до официальных бюллетеней. Редакция должна определить приоритеты по тематике и актуальности, раздать задания корреспондентам на подробные репортажи и фотоматериалы. Регулярно проводится анализ мнения читателей и выбор тем для более сложных материалов: обзор за неделю, журналистское расследование, специальные интервью, сборник советов.

Все многообразие публикуемых в периодических печатных изданиях материалов можно классифицировать как по темам (политика, экономика, региональные новости, общественные проблемы, наука и техника, культура, криминал, светская хроника, спорт), так и по информационной насыщенности (анонс (краткое сообщение в несколько строк), заметка (2–5 абзацев), подробный репортаж и еженедельный обзор). Большое значение для успеха газеты имеет актуальность издаваемого материала по размаху описываемых событий (федеральная, региональная, местная), по времени (в том числе сезонные материалы) и по популярности (рейтинг, анкеты, отзывы читателей). Подробная классификация газетных материалов играет большую роль при проектировании автоматизированной информационной системы.

В описанной далее модели акцент делается на процессы сбора информации, сортировки по приоритетам, группировки в различные выпуски и распространения газеты. Типографские, экономические и правовые аспекты издательской деятельности остаются пока за рамками рассмотрения. В будущем они будут включены в более сложную и более адекватную модель издательства.

## 3. Модель газеты

Основная модель (essential model) согласно методике Йордона [8] состоит из краткого описания выполняемых системой функций и контекстной диаграммы, на которой отображается обмен информацией между системой и объектами окружающего мира. Конечная цель – определить границы моделирования. Например, корреспонденты могут быть штатными сотрудниками газеты, но процесс написания репортажей является творческим и не входит в автоматизированную систему обработки информации.

Рассматриваемая газета получает информацию о предстоящих событиях, проводит её сортировку и группировку, распределяет задания своим корреспондентам, обрабатывает получаемые репортажи, формирует выпуски и учитывает мнения и интересы читателей при отборе материала. Эти действия составляют главное назначение моделируемой системы.

Так как объекты окружающего мира чётко делятся на поставщиков и потребителей новостной информации, то удобно разделить контекстную диаграмму на две части. На первой диаграмме (рис. 1) показаны агенты – источники материалов для формирования выпусков газеты.

Когда ожидается любое локальное событие, описание которого может поднять рейтинг издания, вся надежда издательства на информаторов, будь то информационные агентства калибра Reuters или InterFax, или же одиночки, которые за небольшую плату готовы выложить все секреты мира. Назовём условно слухом сообщение о предстоящем реальном событии, содержащее его место и время (в общем это некий интервал), степень достоверности (вероятность того, что оно произойдёт) и одну или несколько рубрик для классификации, а возможно и предполагаемый заголовок для новостной ленты. Тематические рубрики целесообразно выбирать из заранее созданного иерархического словаря. Хотя очевидно, что задача создания удобной классификации всех событий

невероятно сложна. Например, и биологическая классификация видов и научный классификатор УДК далеки от совершенства.


Рис. 1. Основная модель: Информационные потоки между агентами и газетой

Помимо информаторов для планирования репортажей используются данные официальных структур всех уровней, от которых газета получает расписание публичных мероприятий — данные о планируемых событиях в политической и экономической сферах, а также готовые пресс-релизы о состоявшихся встречах и совещаниях. Сюда же можно отнести и другие источники публичной информации, такие как оргкомитеты научно-практических конференций, сайты спортивных организаций, календари культурных учреждений. С точки зрения нашей модели эти данные приходят регулярно, и к репортажам можно подготовиться заранее. Значит, можно составить календарь, на основе которого планируются репортажи.

Отдел планирования реагирует на поступление слухов и расписаний публичных мероприятий, проверяет их достоверность и отбирает события, имеющие интерес для читателей газеты. Учитывая имеющиеся в распоряжении людские ресурсы, раздаются задания на написание репортажей и получение фотографий с места событий. Корреспонденты и фотокорреспонденты присылают в газету два вида сообщений: заметка (1 фотография) и подробный репортаж с несколькими фотографиями.

Кроме этого, на особо интересные для читателей темы выдаются особые задания для обозревателей и специальных корреспондентов. Их работа включается в расширенный выпуск или отдельное приложение. Это, например, обзор за неделю, журналистское расследование, специальные интервью, сборник советов. Вовремя отпечатанная глава из ещё не опубликованного романа талантливого писателя способна поднять рейтинг газеты. Поэтому в нашей модели отражены и фрагменты художественных произведений. Однако такие вещи на заказ не пишутся, и спланировать их заранее не представляется возможным. Так как экономические аспекты не рассматриваются, то рекламных материалов в нашей модели нет.

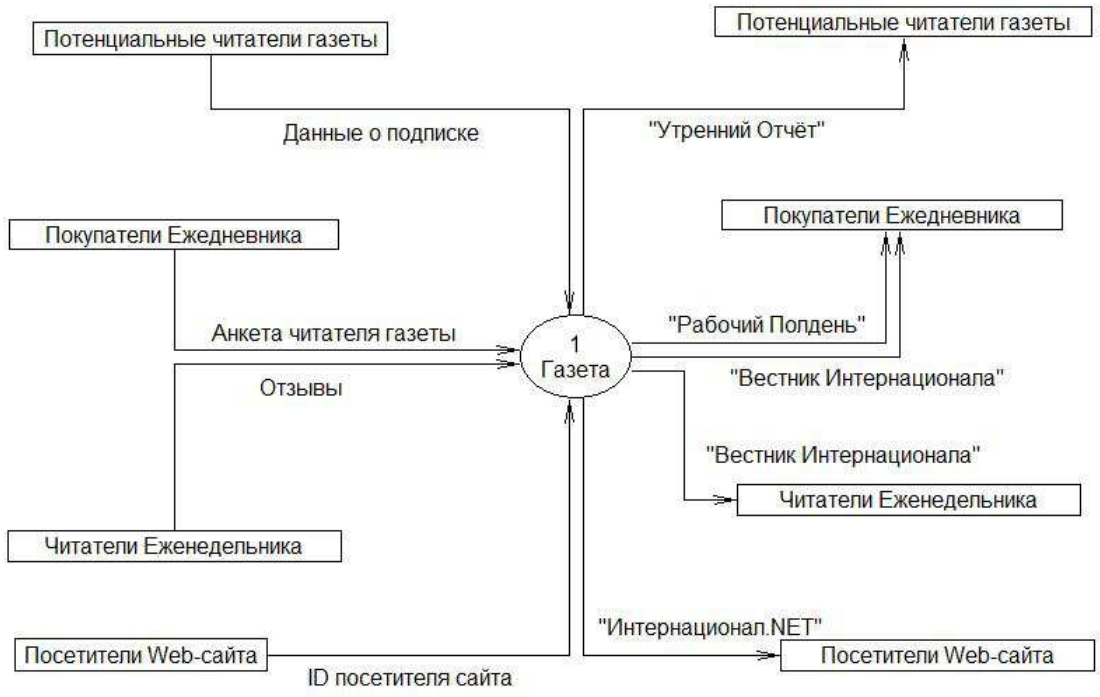


Рис. 2. Основная модель: Информационные потоки между читателями и газетой

На второй контекстной диаграмме (рис. 2) отображены информационные потоки, связывающие газету с читателями. Читатели подразделяются на несколько категорий в зависимости от версии газеты и их активности в обратной связи с редакцией. Распространение утренней газеты среди потенциальных читателей может подтолкнуть некоторых людей к приобретению ежедневного выпуска. В нем для некоторых материалов публикуется адрес (электронный или обычный), по которому читатели ежедневника могут прислать отзыв со своим мнением на затронутую тему. Однако такие случаи редки.

Более активны в обратной связи постоянные читатели, которые получают не только ежедневные, но и еженедельные выпуски, в которых регулярно печатаются подробные анкеты о понравившихся темах и предполагаемой популярности будущих публикаций. Посетители Web-сайта могут оставлять коммен-

тарии на форуме и заполнять электронные анкеты. По информационной сути электронные и печатные отзывы и анкеты неразличимы. Предполагается учитывать активность читателей в конфиденциальной базе данных для возможных привилегий.

Информационные потоки, связывающие внешних агентов с газетой на контекстной диаграмме, состоят из данных с ясной структурой, которая приведена в таблице 1.

Таблица 1. Предварительная структура потоков данных

| Поток данных | Структура |
|---|---|
| Слухи | Источник, достоверность, тема, время, место, заголовок |
| Расписание мероприятий | Тема, источник, список (время, место, заголовок) |
| Пресс-релиз | Источник, тема, время, место, заголовок, текст |
| Задание на фотографии | Исполнитель, приоритет, тема, время, место, заголовок, срок, объем |
| Задание на репортаж | Исполнитель, приоритет, тема, время, место, заголовок, срок, объем |
| Задание на обзор | Исполнитель, приоритет, тема, срок |
| Особое задание | Исполнитель, приоритет, тема, жанр, срок |
| Фотографии | Список(изображение, подпись) |
| Заметка | Тема, заголовок, краткий текст, подпись |
| Подробный репортаж | Тема, заголовок, подробный текст, подпись |
| Обзор | Тема, заголовок, текст обзора, подпись |
| Особые материалы | Тема, заголовок, текст, жанр, подпись |
| Фрагменты произведений | Автор, название, текст, размер текста |
| Отзывы | Заголовок, номер выпуска, текст отзыва, дата |
| Анкета читателя | Номер выпуска, имя, список ответов, дата |

Заметив повторения, вводим новый объект – событие, имеющий атрибуты: тема, место, время, заголовок. Методы этого объекта устанавливают его ассоциации с последующими на него репортажами, фотографиями, обзорами, отзывами и т.п.

## 4. Механизмы реагирования на события

После определения границ моделирования следующий этап – идентификация событий, на которые система должна реагировать, и определение механизмов реагирования на эти события. Список основных событий приведен в таблице 2.

Для облегчения сортировки материалов по темам и приоритетам вводится подсистема «Колонки». В данной подсистеме собранные материалы группируются в макеты новостей. После утверждения из этих макетов верстается го-

товый номер газеты, который отправляется в печать. Обмен информацией с агентами возлагается на подсистему «Канцелярия». В ней происходит сбор и обработка информации от агентов и выдача им заданий. Обмен информацией с читателями и техническая сторона вопроса возлагаются на подсистему «Отдел технической поддержки, распространения и обратной связи».

Таблица 2. Реагирование на внешние события

| Событие | Реакция |
|---|---|
| Получение слуха | 1. Проверить достоверность<br>2. Определить популярность темы<br>3. Назначить приоритет<br>4. Определить исполнителя<br>5. Выдать задание |
| Получение репортажа | 1. Проверить соответствие заданию<br>2. Определить связанные фотоматериалы<br>3. Отправить заметку в утренний выпуск и интернет-версию<br>4. Передать текст в колонки |
| Получение отзыва | 1. Регистрация в рейтинге читателей<br>2. Изменение популярности тем<br>3. Передача текста отзыва в колонки |

Подробные механизмы реагирования на события лучше представить в форме диаграмм деятельности (Activity diagram) на языке UML. Это было сделано с использованием продукта Rational Software Architect, полученного по программе Academic Initiative от фирмы IBM. Однако диаграммы деятельности являются лишь вспомогательным средством уточнения модели, а не средством разработки алгоритмических блок-схем.

Подробная спецификация процессов обработки событий является результатом следующего этапа – объектно-ориентированного проектирования, который будет освещён в следующей статье.

## Заключение

Рассмотренная выше модель информационной системы обработки новостной информации можно использовать при объектно-ориентированном проектировании автоматизированных систем сортировки и группировки сообщений. Область применения не ограничивается газетами (или журналами), но может

включать как продолжающиеся издания, так и чисто электронные интернет-ресурсы, имеющие задачу реагирования на предстоящие события.

## Литература


1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. 2-е изд. М. : Издательство Бином ; СПб. : Невский диалект, 1999.
2. Coad P., Yourdon E. Object-oriented analysis. Prentice Hall, 1990.
3. IDEF1 Information Modeling : Technical Report. AFWAL-TR-81-4023.
4. Jacobson I., Christeron M., Jonsson P., Overgaard G. Object-oriented software engineering. Addison-Wesley, 1992.
5. Рамбо Дж., Блаха М. UML 2.0 Объектно-ориентированное моделирование и разработка. СПб. : Изд-во «Питер», 2007.
6. O'Docherty M. Object-Oriented Analysis and Design. Understanding System Development with UML 2.0. John Wiley & Sons Ltd, 2005.
7. Йордон Э., Аргила К. Объектно-ориентированный анализ и проектирование систем. М. : Изд-во «Лори», 2007.
8. Yourdon E. Just Enough Structured Analysis. Yourdon Press, 2006.
9. Элиенс А. Принципы объектно-ориентированной разработки программ. М. : Изд-во «Вильямс», 2002.
10. Анализ, моделирование и проектирование информационных систем. URL: `http://alice.stup.ac.ru/case/caseinfo` (дата обращения 03.03.2006).

*Математические структуры и моделирование*
2011, вып. 23, с. 58–75

УДК 004.4'24

# ЯЗЫК ДЕТАЛИЗАЦИИ КАРКАСА ПРОГРАММНЫХ КОМПОНЕНТОВ ПОДДЕРЖКИ ЗАНЯТИЙ ЛИНГВИСТИЧЕСКОЙ НАПРАВЛЕННОСТИ

**С.В. Гусс**

Для каркаса программных компонентов поддержки занятий лингвистической направленности предлагается предметно-ориентированный язык детализации, позволяющий повысить эффективность разработки и сопровождения конечной программной системы.

## Введение

В предлагаемой работе речь пойдёт о ***предметно-ориентированном языке*** (DSL – Domain-Specific Language), специфичном для определённой предметной области (домена) и практическим вопросам его развития и применения в реальных проектах. Такие языки создаются в контексте ***предметно-ориентированного проектирования*** (DDD - Domain-Driven Design). Это проектирование представляет собой подход к решению проблем, который применим тогда, когда одна и та же особенная проблема появляется и повторяется вновь и вновь (от одной члена семейства близких по функциональности программных продуктов к другому). С каждым проявлением данной проблемы выявляется её характер и методики управления ею (методика может быть представлена в виде обобщённой модели). Этот подход к проектированию особо эффективен вместе с использованием другого, более общего подхода к разработке приложений, ***подхода с использованием моделей*** (MDD – Model-Driven Development). Он позволяет в процессе проектирования пользоваться терминами той предметной области, для которой разрабатывается приложение, и предполагает создание двух взаимосвязанных компонентов в рамках построения конечного приложения. ***Первый компонент*** – фиксированная часть, которая может быть представлена, в зависимости от масштаба и формы проблемы, каркасом, программной платформой, интерпретатором или каким-нибудь прикладным программным интерфейсом (API). Разрабатывается такой компонент в рамках стандартного подхода к проектированию, в рамках используемой организации процесса разработки (будь то формальный процесс, либо на основе гибких методологий), с ручным кодированием и тестированием (например, с предварительным


Омский государственный университет им. Ф.М. Достоевского
E-mail: InfoGuss@Gmail.com

написанием модульных тестов). ***Второй компонент*** – переменная часть, программный код которой претерпевает изменения при переходе от одного члена программного семейства к другому. Предметно-ориентированный язык работает с переменной частью, учитывая характеристики и специфику фиксированной. Одно из возможных воплощений предметно-ориентированного языка – визуальный инструмент моделирования в рамках программной среды разработки, предлагающий средства составления моделей и способный порождать программный код на основе этих моделей.

В данной работе предлагается описание предметно-ориентированного языка, работающего с каркасом программных компонентов поддержки занятий лингвистической направленности. Язык представлен в рамках так называемого рецепта детализации, где суть применения языка излагается в виде примеров программного кода. Рецепт детализации иногда называется рецептурным справочником и служит альтернативой формальной документации разработчика.

## 1. Предшествующие работы

Представленная в статье информация основана на исследованиях в области проектирования программных средств и составлена на базе ряда предшествующих ей научных и научно-практических работ различного характера. Эти работы близки по тематике к исследованиям автора статьи и тем или иным образом оказали влияние на ход его мысли. Их можно разделить на две категории. К первой категории относятся статьи и отчёты, опубликованные в научно-практических и научно-методических журналах, посвящённых вопросам образования и его автоматизации, методам программной инженерии и новым информационным технологиям. Ко второй категории работ можно отнести научно-популярные издания и монографии, из которых можно почерпнуть ряд идей, принципов, подходов и укоренившихся в практике методов и методик.

К ***первой категории*** можно отнести ряд исследовательских работ автора статьи [1–7]. Эти работы делятся на три группы.

К ***первой группе*** относятся исследования, затрагивающие концептуальные вопросы создания программных систем учебного назначения лингвистической направленности [1, 2]. В этих работах даётся описание области электронного обучения с применением компьютерных игр. С позиции педагога игровые программные системы учебного назначения являются инструментами, которые помогают саморазвитию вовлечённого в процесс ученика и развивают реакцию для поиска решений на поставленную задачу. Их можно использовать как дополнительное средство для контроля знаний, как средство проверки усвоения материала. С позиции разработчика они состоят из ряда подсистем. К этим подсистемам относят подсистему управления пользователями, обучения, обслуживания, взаимодействия с учениками, безопасности, коммуникации, учебного материала, психофизиологического сопровождения. Вся обучающая система в целом имеет своих пользователей. К ним относятся ученик, учитель, методист, администратор и т.д. Они запускаются в рамках специальных технических устройств, таких как рабочие станции и мобильные системы.

Ко ***второй группе*** [2–5] относятся исследования практического характера, имеющие отношение к вопросу создания обучающих игровых систем. В них описаны процессы создания реальных приложений, часть из которых использовалась на практике в процессе обучения и проверки знаний студентов. На базе проведённых исследований был накоплен программный задел повторного использования для будущих разработок. В работах [4, 5] представлена модель (на разных уровнях абстракции) каркаса программных компонентов поддержки занятий лингвистической направленности. В данной же статье представлен предметно-ориентированный язык, который работает с реализованной моделью каркаса.

К ***третьей группе*** относятся работы [6, 7], в которых проводится анализ и обобщение подходов и приёмов разработки, подходящих для предметной области игровых обучающих программных систем.

К первой категории можно также отнести ряд ***близких по тематике работ*** [8–11]. Они представлены исследованиями А. Фуртадо (Andre W.B. Furtado), А. Сантоса (Andre Santos) и другими. Их работы связаны, прежде всего, с исследованием и разработкой предметно-ориентированных языков моделирования, предметно-ориентированным проектированием в целом, а также семейством программных систем. Направление исследований – развитие и адаптация концепции фабрик программного обеспечения для создания игровых программных систем, однако значительный акцент делается на представление игрового процесса в графической форме. К этой же категории можно отнести работы [12, 13] отечественных исследователей А.С. Куракина и О. А. Шаболиной. Они имеют дело с исследованием практической стороны процесса разработки. В первом случае представлен процесс создания системы обучения для корпоративного сектора и проанализирован ряд проблем, встающих на этом пути. Во втором случае представлено альтернативное использование обучающих систем, предлагается обучать учеников методам и процессам создания программных средств на основе разработки игровых программных систем. Нечто подобное представлено также в работе [14], где в процессе создания компьютерных игр предлагается развивать не только программистские навыки.

***Вторая категория*** представлена работами [15–24] общего характера, содержащими описание принципов и методик проектирования и разработки программных средств. В работах [15–17] представлено описание процессов и этапов жизненного цикла программных систем. В работе [15] излагаются основы гибких технологий моделирования (Agile modeling), их применение в рамках организации процессов экстремального подхода (eXtreme Programming) к программированию и унифицированного подхода. Работа [16] устанавливает связь между практически ставшим стандартом в разработке объектно-ориентированных систем унифицированным языком моделирования (UML – Unified Modeling Language) и унифицированным процессом (UP – Unified Process). Работа [17] отечественного исследователя В.В. Липаева характеризует инженерный подход к созданию программных средств, в котором учитываются и оцениваются такие характеристики, как трудоёмкость, длительность, производство на единицу размера программной продукции, средняя производи-

тельность труда, прототипы проектов, сложность, размер, наличие повторно-используемых компонентов и т.д. Работы [18–24] связаны с такими направлениями, как предметно-ориентированное проектирование и моделирование, повторное использование компонентов программных систем. Особо следует выделить работу Э. Эванса (Eric Evans), предлагающего специалисту предметной области и инженеру-разработчику «единый язык, на котором они могли бы разговаривать друг с другом» [22].

## 2. Модель предметной области

### 2.1. Каркас

**Диаграмма классов** каркаса представлена на рис. 1.

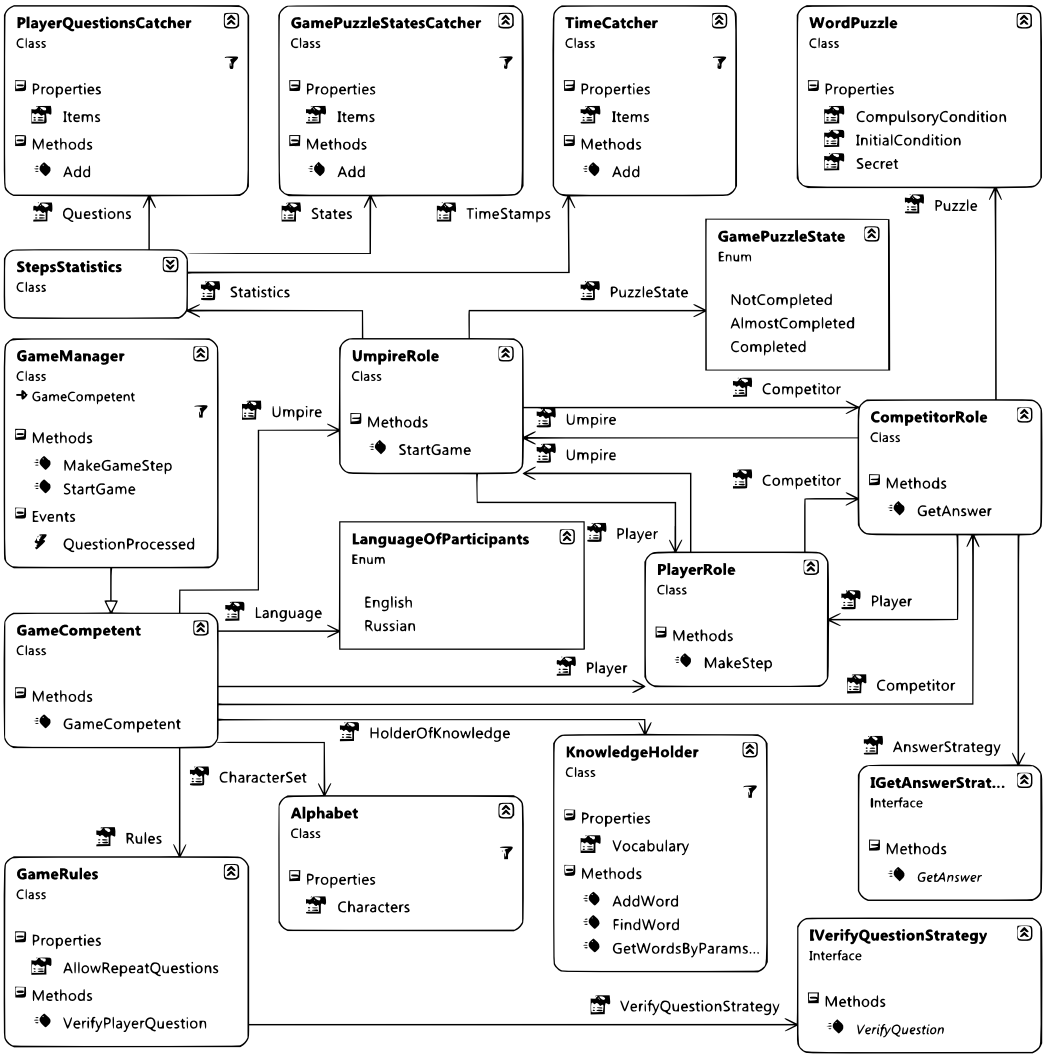


Рис. 1. Диаграмма классов каркаса

**Основные члены каркаса** представлены в следующих колонках. Использу-

емые термины предметной области (взяты в кавычки) разъясняются в разделе «Словарь предметной области» далее в статье.

Пространство имён **GSEducation.AQLPGFramework**:

| **Класс** | **Составляющие** |
|---|---|
| public class Alphabet – работа с «алфавитом» заданного «языка»: | – public Alphabet(LanguageOfParticipants language, bool useUpperCase) – конструктор, необходимо задать требуемый «язык» и указать тип выдачи литер – в верхнем или нижнем регистре;<br>– public List<string> Characters {get;} – элемент доступа к «алфавиту» «языка»; |
| public class CompetitorRole – работа с «ролью» «соперника» | – public GamePuzzleState GetAnswer(string[] question, out string answer, GameRules gameRules) - получить ответ от «соперника», хранящего «задачу-головоломку» на «вопрос» «игрока». Выполняется логикой каркаса во время вызова метода PlayerRole.MakeStep();<br>– public IGetAnswerStrategy AnswerStrategy {set; get;} – интерфейс класса, реализующего стратегию получения «ответа» на «вопрос»;<br>– public PlayerRole Player {set; get;} – «игрок», ассоциированный с «соперником»;<br>– public WordPuzzle Puzzle {set; get;} – лингвистическая «задача-головоломка»;<br>– public UmpireRole Umpire {set; get;} – «судья», ассоциированный с «соперником»; |
| public class GameCompetent – класс объектов, имеющих доступ к управлению элементами «процесса решения» «задачи-головоломки» | – public GameCompetent(LanguageOfParticipants language) – конструктор, требует задания языка;<br>– public Alphabet CharacterSet {set; get;} – набор «литер» алфавита;<br>– public CompetitorRole Competitor {set; get;} – доступ к «сопернику»;<br>– public KnowledgeHolder HolderOfKnowledge {set; get;} – доступ к «лексемам» языка;<br>– public LanguageOfParticipants Language {set; get;} – язык текущего процесса;<br>– public PlayerRole Player {set; get;} – «игрок» «процесса решения»;<br>– public GameRules Rules {set; get;} – «правила» «процесса решения»;<br>– public UmpireRole Umpire {set; get;} – «судья» «процесса решения»; |
| public class GameManager : GameCompetent – класс объектов, управляющих «процессом решения» | – public void StartGame() – начать процесс;<br>– public bool MakeGameStep(string[] question) – сделать «шаг» в «процессе решения» «задачи-головоломки»;<br>– public event EventHandler<QuestionProcessedEventArgs> QuestionProcessed – событие, на которое необходимо подписаться, чтобы получать «ответы» на «вопросы»;<br>Примечание: именно эти три составляющих представляют интерес для разработчика приложений из готовых компонентов, полученных путём детализации каркаса; |
| public enum GameProcessState – «состояние процесса решения» «задачи-головоломки» | – Active – процесс активен (уже начался, но ещё не завершён);<br>– Inactive – процесс не активен (либо ещё не начался, либо уже завершён); |
| public enum GamePuzzleState – «состояние решения задачи». Имеет место в контексте метода GetAnswerLogic класса CustomizableGetAnswerStrategy (листинг 2, далее по тексту) | – AlmostCompleted – задача почти решена;<br>– Completed – задача решена полностью;<br>– NotCompleted – задача не решена (возможно, ещё и не решалась); |
| public enum LanguageOfParticipants – заданный «язык» | – English – английский язык;<br>– Russian – русский язык; |
| public enum QuestionVerificationResult – «результат обработки вопроса». Имеет место в контексте метода VerifyQuestionLogic класса CustomizableVerifyQuestionStrategy (листинг 2, далее по тексту) | – Correct – вопрос-утверждение участника соответствует правилам задачи, действующим в рамках процесса;<br>– Incorrect – не соответствует правилам либо повторяется в то время, когда повторы запрещены;<br>– Repeat – соответствует правилам задачи, но уже имело место в рамках процесса;<br>Примечание: при инициализации класса GameManager можно указать - разрешены «повторения вопросов» со стороны главного участника или запрещены; |

| | |
|---|---|
| public class GamePuzzleStatesCatcher - хранение всех «состояний решения задачи», полученных за процесс (доступны через GameCompetent.Umpire.Statistics.States) | – public void Add(GamePuzzleState state) – добавить состояние (выполняется автомтически логикой каркаса);<br>– public List<GamePuzzleState> Items {get;} – элементы всех «состояний решения задачи», учитываются автоматически логикой каркаса; |
| public class GameRules - правила «задачи-головоломки», которые действуют во время процесса | – public QuestionVerificationResult VerifyPlayerQuestion(string[] question, UmpireRole umpire) – проверить «вопрос» «игрока»;<br>– public bool AllowRepeatQuestions {set; get;} – разрешение или запрет на повторные «вопросы» со стороны «игрока»;<br>– public IVerifyQuestionStrategy VerifyQuestionStrategy {set; get;} – стратегия проверки утверждений со стороны игрока; |
| public class KnowledgeHolder – база знаний, содержащая словарные лексемы языка | – public KnowledgeHolder(LanguageOfParticipants language) – конструктор, требует указания «языка» для работы;<br>– public void AddWord(string word) – добавить «лексему» в базу знаний на период «процесса решения»;<br>– public bool FindWord(string word) – проверить наличие конкретной «лексемы» в базе;<br>– public List<string> GetWordsByParams(int length) – найти все «лексемы» в базе заданной длины;<br>– public List<string> GetWordsByParams(int length, char firstLetter) – найти все «лексемы» в базе заданной длины, начинающиеся с определённой литеры;<br>– public List<string> Vocabulary {get;} – доступ к «словарю», содержащему «лексемы» текущего языка; |
| public class PlayerQuestionsCatcher – хранитель вопросов, поступающих от игрока | – public void Add(string[] question) – добавить «вопрос», учитываются автоматически логикой каркаса;<br>– public List<string> Items {get;} – получить список «вопросов»; |
| public class PlayerRole - работа с «ролью» «игрока» | – public GamePuzzleState MakeStep(string[] question, out string answer, GameRules gameRules) – сделать «ход» (выполняется логикой каркаса во время вызова метода GameManager.MakeGameStep(..));<br>– public CompetitorRole Competitor {set; get;} – «соперник», ассоциированный с «игроком»;<br>– public UmpireRole Umpire {set; get;} – «судья», ассоциированный с «игроком»; |
| public class QuestionProcessedEventArgs : System.EventArgs – аргументы, передаваемые обработчику событий типа QuestionProcessed | – public QuestionProcessedEventArgs(GamePuzzleState state, string answer) – конструктор, требует указания состояния решения задачи и ответ на вопрос со стороны игрока (выполняется автоматически логикой каркаса во время вызова метода GameManager.MakeGameStep(..));<br>– public string Answer {get;} – «ответ» на «вопрос»;<br>– public GamePuzzleState State {get;} – «состояние решения задачи»; |
| public class StepsStatistics – статистика совершённых ходов | – public PlayerQuestionsCatcher Questions {set; get;} – «вопросы», заданные «игроком» в «процессе решения» «задачи-головоломки»;<br>– public GamePuzzleStatesCatcher States {set; get;} – «состояния решения задачи»;<br>– public TimeCatcher TimeStamps {set; get;} – «временные отметки» совершённых «игроком» «ходов»; |
| public class TimeCatcher - «временные отметки» совершения «ходов» | – public void Add() – добавить временную отметку, учитываются логикой каркаса;<br>– public List<DateTime> Items {get;} – набор «временных отметок» совершения «ходов»; |
| public class UmpireRole – работа с «ролью» «судьи» | – public void StartGame(PlayerRole player, CompetitorRole competitor) – начать «процесс решения» (выполняется автоматически при вызове метода GameManager.StartGame());<br>– public CompetitorRole Competitor {set; get;} – ассоциированы с «судьёй» «соперник»;<br>– public PlayerRole Player {set; get;} – ассоциированный с «судьёй» «игрок»;<br>– public GamePuzzleState PuzzleState {set; get;} – текущее состояние решения задачи;<br>– public StepsStatistics Statistics {set; get;} – «статистика шагов»; |

| | |
|---|---|
| public class WordPuzzle – работа с «задачей-головоломкой» | – public string CompulsoryCondition {set; get;} – «обязательное условие» для решения задачи;<br>– public string InitialCondition {set; get;} – «начальное условие» для решения задачи;<br>– public string Secret {set; get;} – секрет «задачи»; |
| public class WrongPlayerQuestionException : System.Exception – исключения, вызванные некорректным составлением «вопросов» | Конструкторы:<br>– public WrongPlayerQuestionException();<br>– public WrongPlayerQuestionException(string message);<br>– public WrongPlayerQuestionException(string message, System.Exception innerException);<br>– protected WrongPlayerQuestionException(SerializationInfo info, StreamingContext context); |

## 2.2. Шаблон детализации

**Листинг 1. Код шаблона проекта**

```
using System;
using GSEducation.AQLPGFramework;

namespace GSEducation.AQLPGFramework.DetalizationTemplate{

public partial class TemplateGameManager: GameManager{
==================================
public TemplateGameManager()
: base(LanguageOfParticipants.Russian, true){
InitializeGame();}

public TemplateGameManager(LanguageOfParticipants language, bool allowRepeat)
: base(language, allowRepeat){
InitializeGame();}

private void InitializeGame(){
MakePuzzle();}

protected virtual void MakePuzzle(){
throw new NotImplementedException();}}

public partial class TemplateGetAnswerStrategy : IGetAnswerStrategy{
====================================
public TemplateGetAnswerStrategy(GameManager manager){
gameManager = manager;}

protected GameManager gameManager;

public GamePuzzleState GetAnswer(string[] question, out string answer){
GamePuzzleState puzzleState = GamePuzzleState.NotCompleted;
answer = "";
puzzleState = GetAnswerLogic(question, out answer);
return puzzleState;}

protected virtual GamePuzzleState GetAnswerLogic(string[] question, out string answer){
throw new NotImplementedException();}}

public partial class TemplateVerifyQuestionStrategy : IVerifyQuestionStrategy{
========================================
public TemplateVerifyQuestionStrategy(GameManager manager){
gameManager = manager;}

protected GameManager gameManager;

public QuestionVerificationResult VerifyQuestion(string[] question){
QuestionVerificationResult result =
QuestionVerificationResult.Correct;
VerifyQuestionLogic(question, out result);
return result;}
```

```
protected virtual void VerifyQuestionLogic(string[] question, out result){
throw new NotImplementedException();}}}
```

**Листинг 2. Предоставление точек расширения**

```
using System;
using GSEducation.AQLPGFramework;

namespace GSEducation.AQLPGFramework.DetalizationTemplate{
public class CustomizedGameManager : TemplateGameManager{
=========================
public CustomizedGameManager(LanguageOfParticipants language, bool allowRepeat)
: base(language, allowRepeat){}

public CustomizedGameManager()
: base(){}

protected override void MakePuzzle(){
Rules.VerifyQuestionStrategy = new CustomizableVerifyQuestionStrategy(this);
Competitor.AnswerStrategy = new CustomizableGetAnswerStrategy(this);
//TODO:
//==============================
VARIABLE PART - EXTENSION POINT
//==============================
}

private class CustomizableVerifyQuestionStrategy : TemplateVerifyQuestionStrategy{
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
public CustomizableGetAnswerStrategy(GameManager manager)
: base(manager){}

protected override void VerifyQuestionLogic(string[] question,
out QuestionVerificationResult verificationResult){
verificationResult = QuestionVerificationResult.Incorrect;
//TODO:
//==============================
VARIABLE PART - EXTENSION POINT
//==============================
}}

private class CustomizableGetAnswerStrategy : TemplateGetAnswerStrategy{
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
public CustomizableGetAnswerStrategy(UmpireRole umpire, CompetitorRole competitor)
: base(umpire, competitor){}

protected override GamePuzzleState GetAnswerLogic(string[] question, out string answer){
GamePuzzleState puzzleState = GamePuzzleState.NotCompleted;
answer = "";
//TODO:
//==============================
VARIABLE PART - EXTENSION POINT
//==============================
return puzzleState;}}}}
```

Места в листингах, обозначенные как «VARIABLE PART», представляют собой точки расширения, в которые разработчик может вставлять изменяемый код (активно пользуясь элементами каркаса). Листинг 1 содержит код шаблона проекта, который абсолютно не требует изменений, он не содержит точек расширений и служит своеобразным проектным решением, способствующим более комфортной работе по созданию конечного компонента на основе каркаса. В листинге 2 следует изменить название класса «CustomizedGameManager» на более подходящее в конкретном контексте имя. Для предметно-ориентированного языка каркас, короткое описание которого

дано на рис. 1 и в описании членов пространства имён, а также шаблон его детализации (представленный листингами 1 – 2) представляют фиксированную часть.

**Словарь предметной области**

Каркас хорошо подходит для создания компонентов игровых лингвистических задач типа «вопрос-ответ». В нём содержится ряд элементов, реализующих концепции предметной области, перечисленных далее.

***Словарь предметной области*** состоит из следующих элементов.

***Язык*** (LanguageOfParticipants). Представленная реализация каркаса предполагает выбор между английским и русским языком. ***Алфавит*** (Alphabet). Набор литер выбранного языка. ***Лексемы***. Элементы словаря (KnowledgeHolder.Vocabulary) могут быть отфильтрованы *по длине* либо *по начальной литере*. ***Роль***. Определённое представление поведения в процессе решения задачи. Виды ролей: *соперник* (CompetitorRole), *игрок* (PlayerRole), *судья* (UmpireRole). Действия, производимые методами StartGame() и MakeGameStep() класса GameManager, происходят от лица игрока. ***Задача-головоломка*** (WordPuzzle). Состоит из *обязательного условия*, *начального условия* и *секрета*. ***Обязательное условие*** (CompulsoryCondition) – то, без чего невозможно решение задачи. Например, это может быть требование – составлять лексемы, начинающиеся только на определённую букву, определённой длины или содержащие заданные буквы. В реализации каркаса этот элемент представлен в виде строковой переменной, формат и логика обработки которой должна предусматриваться создателем задачи. Начальное условие и секрет также представлены в виде строковых переменных и соответственно предполагают создания логики их обработки и интерпретации. ***Начальное условие*** (InitialCondition) – то, с чего следует начинать процесс решения. Например, это может быть требование – начинать процесс с составления лексемы определённой длины. ***Ответ*** – действие соперника по отношению к игроку. ***Вопрос*** – действие игрока по отношению к сопернику. Ответы и вопросы — это элементы диалога, сообщения, которыми они обмениваются. В листинге 2, в методах VerifyQuestionLogic() и GetAnswerLogic() представлены точки расширения, которые требуют определения логик проверки вопросов и выдачи соответствующего ответа. В рамках логики проверки вопросов (VerifyQuestionLogic()) необходимо выдать результат обработки вопроса (QuestionVerificationResult). Результат может быть следующим: *корректный вопрос*, *некорректный вопрос*, *повторный вопрос*. В рамках логики ответа на вопрос (GetAnswerLogic()) необходимо указать состояние решения задачи (GamePuzzleState). Состояния решения задачи: *почти решена*, *решена*, *не решена*. ***Процесс решения***. Состояние процесса решения: *активный*, *неактивный*. Если задача решена полностью, логика каркаса переводит процесс в неактивное состояние. ***Правила*** (GameRules). Следят за исполнением логики проверки вопроса (VerifyQuestionLogic). ***Шаг***. В рамках процесса решения задачи шаг представляет собой последовательность действий — задать вопрос, получить ответ. ***Статистика шагов*** (StepsStatistics). Логика каркаса фик-

сирует каждый шаг в рамках статистики. Статистика учитывает *состояния решения задачи*, *временные отметки*, *вопросы*.

### 2.3. Примеры детализации каркаса

**Лингвистическая задача «What Letter I Thought» (рис. 2)**

***Описание задачи***. Условие – случайным образом загадывается буква, которую необходимо отгадать игроку. Игра состоит из двух этапов. На первом этапе игрока просят последовательно называть слова, в которых предположительно содержится загаданная буква. Если при очередном вопросе игрока, выраженном в виде требуемого слова, обнаружится, что это слово содержит загаданную букву, игрок переводится на следующий этап. На этом этапе он должен задавать вопросы в виде букв. Т.е. первый этап завершается тогда, когда игрок находит слово, в котором есть загаданная буква. Теперь на основе этого игрок должен найти букву. Когда она будет найдена, задача будет решена. Представление игрового процесса может быть оформлено интерфейсом, приведённым на рис. 2 (именно так оформлен игровой процесс одного из игровых продуктов автора статьи, проекта под названием «We Need Your Help», в котором используются компоненты, созданные путём детализации представленного каркаса). В верхней части окна графического интерфейса содержится информация в виде общих требований к игроку, того, что ему необходимо сделать, чтобы решить задачу. В левой части — требования и указания, касающиеся конкретного этапа процесса. Справа — отображение диалога игрока с системой в виде вопросов и ответов.

***Логика создания задачи-головоломки*** (точка расширения MakePuzzle() в листинге 2) заключается в следующем. Необходимо выбрать случайным образом букву, которую игрок должен угадать. В данном случае можно воспользоваться GameCompetent.CharacterSet.Characters и CompetitorRole.Puzzle.Secret элементами предметной области (Листинг 3). Случайным образом выбирается буква из алфавита текущего языка. Её значение становится секретом задачи-головоломки.

***Логика стратегии проверки вопроса игрока*** (точка расширения метода VerifyQuestionLogic(..)) представлена в листинге 3. После того как игрок сделал ход, задал вопрос, предложив слово, происходит его проверка в соответствии с текущим этапом игры. Если состояние решения задачи на данный момент – ещё не решена, значит, сейчас первый этап, если – почти решена, то второй. На первом этапе вопрос проверяется по словарю, на втором – по алфавиту. Выносится соответствующее решение о корректности или некорректности заданного вопроса. Как было сказано ранее в описании каркаса, вопрос может задаваться повторно. Повторения можно разрешить или запретить. Решить этот вопрос с запретом предлагается разработчику конечного предложения, который должен указать в конструкторе класса, представляющего данную игру в готовом к применению компоненте, запрет или разрешение повторов. Сами же повторы будут отслеживаться автоматически или вообще не отслеживаться логикой каркаса, в зависимости от сделанного выбора (разрешение или запрет).

***Логика стратегии выдачи ответа*** (точка расширения GetAnswerLogic(..)) представлена в листинге 3. Суть такова – анализируется вопрос игрока. Если его вопрос, он же требуемое слово (либо буква, если игрок на втором этапе), содержит секрет задачи, т.е. букву, которую нужно угадать, то в зависимости от того, на каком этапе находится игрок, решается, перевести его на второй этап или же поздравить с победой. На первый взгляд кажется, что игрок на первом же этапе может назвать требуемую букву, не переходя на второй этап. Однако это невозможно, т.к. вышеописанная логика стратегии проверки вопроса игрока на первом этапе сравнивает его вопрос на принадлежность словарю. А словарь, по крайней мере входящий в реализацию описываемого каркаса, не содержит слов длиною в одну букву.


Рис. 2. Лингвистическая задача «What Letter I Thought»

**Листинг 3. Лингвистическая задача «What Letter I Thought»**

**MakePuzzle() extension point:**

```
Competitor.Puzzle = new WordPuzzle();
Random randomizer = new Random();
List<string> alphabet = CharacterSet.Characters;
int currentIndex = randomizer.Next(0, alphabet.Count - 1);
string randomizerAlphabetEffect = alphabet[currentIndex];
Competitor.Puzzle.Secret = randomizerAlphabetEffect;
```

**VerifyQuestionLogic(..) extension point:**

```
if (gameManager.Umpire.PuzzleState == GamePuzzleState.NotCompleted){
if (!gameManager.HolderOfKnowledge.Vocabulary.Contains(question[0])){
verificationResult = QuestionVerificationResult.Incorrect; return;}}
if (gameManager.Umpire.PuzzleState == GamePuzzleState.AlmostCompleted){
if (question[0].Length == 1 && !gameManager.CharacterSet.Characters.Contains(question[0])){
verificationResult = QuestionVerificationResult.Incorrect;
return;}}
```

**GetAnswerLogic(..) extension point:**

```
if (question[0].Contains(gameManager.Competitor.Puzzle.Secret)){
```

```
if (gameManager.Umpire.PuzzleState == GamePuzzleState.NotCompleted){
puzzleState = GamePuzzleState.AlmostCompleted;}
else if (gameManager.Umpire.PuzzleState == GamePuzzleState.AlmostCompleted){
puzzleState = GamePuzzleState.Completed;}}
if (!question[0].Contains(gameManager.Competitor.Puzzle.Secret)){
if (gameManager.Umpire.PuzzleState ==
GamePuzzleState.AlmostCompleted && question[0].Length == 1){
return GamePuzzleState.AlmostCompleted;}}}
```

**Лингвистическая задача «What Word I Thought» (рис. 3)**

***Описание задачи***. Условие – случайным образом загадывается слово. От игрока требуется угадать это слово. Игрок последовательно должен задавать вопросы в виде требуемых слов. После каждого вопроса игрок получает ответ. Ответ – количество букв в загаданном слове, которые встречаются в вопросе игрока. То есть ответ представлен в виде подсказки, уточняющей правильность направлений поиска игрока в процессе решения задачи. Задача считается решённой только тогда, когда игрок отгадает загаданное слово. Детализация каркаса представлена в листинге 4.

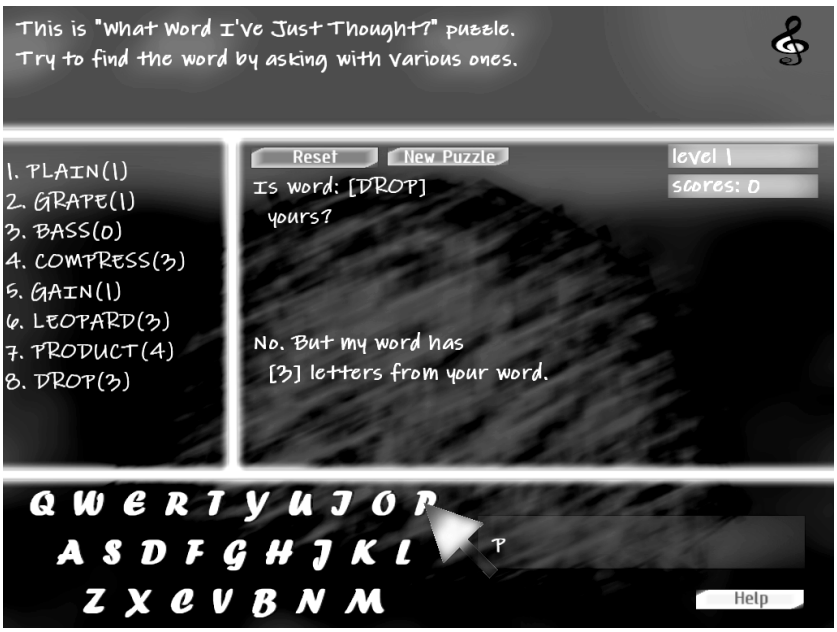


Рис. 3. Лингвистическая задача «What Word I Thought»

**Листинг 4. Лингвистическая задача «What Word I Thought»**

**MakePuzzle() extension point:**

```
Competitor.Puzzle = new WordPuzzle();
Random randomizer = new Random();
List<string> vocabulary = HolderOfKnowledge.Vocabulary;
int currentIndex = randomizer.Next(0, vocabulary.Count - 1);
string randomizerVocabularyEffect = vocabulary[currentIndex];
Competitor.Puzzle.Secret = randomizerVocabularyEffect;
```

**VerifyQuestionLogic(..) extension point:**

```
if (!gameManager.HolderOfKnowledge.Vocabulary.Contains(question[0])){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
```

```
GetAnswerLogic(..) extension point:
if (question[0].Contains(gameManager.Umpire.Competitor.Puzzle.Secret)){
return GamePuzzleState.Completed;}
if (!question[0].Contains(gameManager.Umpire.Competitor.Puzzle.Secret)){
int lettersNumber = 0;
for (int i = 0; i < question[0].Length; i++){
if (gameManager.Umpire.Competitor.Puzzle.Secret.Contains(question[0][i].ToString()))
lettersNumber++;}
answer = lettersNumber.ToString();
return GamePuzzleState.NotCompleted;}
```

### Лингвистическая задача «Funny Stairs»

***Описание задачи***. Реализация алгоритма (включает проверку на возможность решения задачи; если после 10 попыток не удаётся составить задачу, поддающуюся решению, работа алгоритма заканчивается нулевым результатом) представлена в листинге 5. Условия следующие. Обязательное условие: все вопросы игрока должны начинаться с определённой буквы. Начальное условие - количество необходимых для составления игроком слов, выбираемое случайным образом, в данном случае – от 4 до 8. Ещё один момент, который реализован стратегией проверки вопроса, – первый вопрос игрока должен начинаться со слова длиной не более 3. Каждый последующий корректный вопрос должен быть на единицу больше предыдущего. Задача будет решена тогда, когда игрок доберётся до самого последнего вопроса. Таким образом, в начале процесса игроку даются условия – построить лестницу слов, начиная со слов длиною не более 3, причём каждое должно начинаться с определённой обязательным условием буквы. В представленной реализации алгоритма каждый новый вопрос игрока – дополненный предыдущий. Отсюда следует, что игрок может заменять уже составленные в предыдущем ходе слова на новые, более ценные с его точки зрения или точки зрения игровой системы, где используется данный компонент. Например, каркас не предоставляет возможности оценивания результата решения задачи, т.к. было сделано предположение, что это обязанность «игрового движка» (в англоязычной литературе – game engine). Игровой движок может обращаться к статистике решения задачи посредством объекта класса StepsStatistics и произвести оценку в соответствии со своими потребностями.

**Листинг 5. Лингвистическая задача «Funny Stairs»**

```
MakePuzzle() extension point:
Competitor.Puzzle = new WordPuzzle();
Random randomizer = new Random();
List<string> alphabet = CharacterSet.Characters;
bool testResultIsGood = true;
int verificationIterationNumber = 0;
int maxVerificationIterationNumber = 10;
again:
int currentIndex = randomizer.Next(0, alphabet.Count - 1);
string randomizerAlphabetEffect = alphabet[currentIndex];
Competitor.Puzzle.CompulsoryCondition = randomizerAlphabetEffect;
int puzzleSize = randomizer.Next(4, 8);
string randomizerPuzzleEffect = puzzleSize.ToString();
Competitor.Puzzle.InitialCondition = randomizerPuzzleEffect;
List<string> vocabularyItems;
int vocabularyItemLength = 3;
int iterationNumber = Convert.ToInt32(Competitor.Puzzle.InitialCondition);
for (int i = 0; i < iterationNumber; i++){
```

```
vocabularyItems =
HolderOfKnowledge.GetWordsByParams(i + vocabularyItemLength,
Competitor.Puzzle.CompulsoryCondition[0]);
if (vocabularyItems == null || vocabularyItems.Count == 0)
testResultIsGood = false;}
if (!testResultIsGood && verificationIterationNumber <= maxVerificationIterationNumber){
verificationIterationNumber++;
testResultIsGood = true;
goto again;}
else if (!testResultIsGood && verificationIterationNumber > maxVerificationIterationNumber)
Competitor.Puzzle = null;
```

**VerifyQuestionLogic(..) extension point:**

```
int lengthInitialCondition = Convert.ToInt32(
gameManager.Umpire.Competitor.Puzzle.InitialCondition);
if (question.Length > lengthInitialCondition){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
for (int i = 0; i < question.Length; i++){
if (!gameManager.HolderOfKnowledge.Vocabulary.Contains(question[i])){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
if (question[i][0].ToString() != gameManager.Umpire.Competitor.Puzzle.CompulsoryCondition){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
if ((i == 0) && question[0].Length > 3){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
if (i >= 1){
if (question[i].Length - question[i - 1].Length != 1){
verificationResult = QuestionVerificationResult.Incorrect;
return;}}}
```

**GetAnswerLogic(..) extension point:**

```
int lengthInitialCondition = Convert.ToInt32(
gameManager.Umpire.Competitor.Puzzle.InitialCondition);
if (question.Length == lengthInitialCondition)
return GamePuzzleState.Completed;
```

### Лингвистическая задача «Word Group»

***Описание задачи***. Алгоритм представлен (включая проверку на возможность решения составленной задачи) в листинге 6. Необходимое условие – случайно выбранное слово из словаря. Начальное условие – длина этого слова, в контексте задачи – количество слов, которые необходимо составить. Длина составляемых игроком слов – произвольная. Главное требование (в соответствии с необходимым условием) – чтобы игрок на каждую букву заданного слова составил по одному своему слову. Составление слов должно идти последовательно, от начальной буквы заданного слова до последней. Ещё один момент, на который необходимо обратить внимание, – логика каркаса не пропустит слов в вопросе игрока, которых нет в словаре каркаса. Чтобы каркас не отвергал слов, которые он не знает, необходимо добавить в его базу знаний новые слова, которые будут действовать только на время процесса решения задачи. Делается это с помощью метода AddWord(..) класса HolderOfKnowledge.

**Листинг 6. Лингвистическая задача «Word Group»** **MakePuzzle() extension point:**

```
Competitor.Puzzle = new WordPuzzle();
Random randomizer = new Random();
List<string> vocabulary = HolderOfKnowledge.Vocabulary;
bool testResultIsGood = true;
```

```
int verificationIterationNumber = 0;
int maxVerificationIterationNumber = 10;
again:
int currentIndex = randomizer.Next(0, vocabulary.Count - 1);
string randomizerVocabularyEffect = vocabulary[currentIndex];
Competitor.Puzzle.CompulsoryCondition = randomizerVocabularyEffect;
string randomizerPuzzleEffect =
Competitor.Puzzle.CompulsoryCondition.Length.ToString();
Competitor.Puzzle.InitialCondition = randomizerPuzzleEffect;
List<string> vocabularyItems = new List<string>();
int vocabularyItemLength = Convert.ToInt32(Competitor.Puzzle.InitialCondition);
int iterationNumber = Convert.ToInt32(Competitor.Puzzle.InitialCondition);
for (int i = 0; i < iterationNumber; i++){
vocabularyItems = HolderOfKnowledge.GetWordsByParams(
vocabularyItemLength,
Competitor.Puzzle.CompulsoryCondition[i]);
if (vocabularyItems == null || vocabularyItems.Count == 0)
testResultIsGood = false;}
if (!testResultIsGood && verificationIterationNumber <= maxVerificationIterationNumber){
verificationIterationNumber++;
testResultIsGood = true;
goto again;}
else if (!testResultIsGood && verificationIterationNumber > maxVerificationIterationNumber)
Competitor.Puzzle = null;
```

**VerifyQuestionLogic(..) extension point:**

```
int lengthInitialCondition = Convert.ToInt32(
gameManager.Umpire.Competitor.Puzzle.InitialCondition);
if (question.Length > lengthInitialCondition){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
for (int i = 0; i < question.Length; i++){
if (!gameManager.HolderOfKnowledge.Vocabulary.Contains(question[i])){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
if (question[i][0] != gameManager.Umpire.Competitor.Puzzle.CompulsoryCondition[i]){
verificationResult = QuestionVerificationResult.Incorrect;
return;}}
```

**GetAnswerLogic(..) extension point:**

```
int lengthInitialCondition = Convert.ToInt32(
gameManager.Umpire.Competitor.Puzzle.InitialCondition);
if (question.Length == lengthInitialCondition)
return GamePuzzleState.Completed;
```

### Лингвистическая задача «Word Square»

***Описание задачи***. Описание алгоритма представлено в листинге 7. Отличие от предыдущего случая - длины всех составляемых игроком слов должны быть одного размера.

**Листинг 7. Лингвистическая задача «Word Square»**

**MakePuzzle() extension point:**

```
Competitor.Puzzle = new WordPuzzle();
Random randomizer = new Random();
List<string> vocabulary = HolderOfKnowledge.Vocabulary;
bool testResultIsGood = true;
int verificationIterationNumber = 0;
int maxVerificationIterationNumber = 10;
again:
int currentIndex = randomizer.Next(0, vocabulary.Count - 1);
string randomizerVocabularyEffect = vocabulary[currentIndex];
Competitor.Puzzle.CompulsoryCondition = randomizerVocabularyEffect;
string randomizerPuzzleEffect =
Competitor.Puzzle.CompulsoryCondition.Length.ToString();
Competitor.Puzzle.InitialCondition = randomizerPuzzleEffect;
```

```
List<string> vocabularyItems = new List<string>();
int vocabularyItemLength = Convert.ToInt32(Competitor.Puzzle.InitialCondition);
int iterationNumber = Convert.ToInt32(Competitor.Puzzle.InitialCondition);
for (int i = 0; i < iterationNumber; i++){
vocabularyItems = HolderOfKnowledge.GetWordsByParams(vocabularyItemLength,
Competitor.Puzzle.CompulsoryCondition[i]);
if (vocabularyItems == null || vocabularyItems.Count == 0)
testResultIsGood = false;}
if (!testResultIsGood && verificationIterationNumber <= maxVerificationIterationNumber){
verificationIterationNumber++;
testResultIsGood = true;
goto again;}
else if (!testResultIsGood && verificationIterationNumber > maxVerificationIterationNumber)
Competitor.Puzzle = null;
```

**VerifyQuestionLogic(..) extension point:**

```
int lengthInitialCondition = Convert.ToInt32(
gameManager.Umpire.Competitor.Puzzle.InitialCondition);
if (question.Length > lengthInitialCondition){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
for (int i = 0; i < question.Length; i++){
if (!gameManager.HolderOfKnowledge.Vocabulary.Contains(question[i])){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
if (question[i][0] != gameManager.Umpire.Competitor.Puzzle.CompulsoryCondition[i]){
verificationResult = QuestionVerificationResult.Incorrect;
return;}
if ((i >= 1) && (question[i].Length != question[i - 1].Length) &&
(question[i].Length != lengthInitialCondition)){
verificationResult = QuestionVerificationResult.Incorrect;
return;}}
```

**GetAnswerLogic(..) extension point:**

```
int lengthInitialCondition = Convert.ToInt32(
gameManager.Umpire.Competitor.Puzzle.InitialCondition);
if (question.Length == lengthInitialCondition)
return GamePuzzleState.Completed;
```

## Заключение

В работе представлено описание работы предметно-ориентированного языка. Проведён обзор литературы, лежащей в основании исследования автора, указана связь предшествующих работ с работой, представленной в предлагаемой статье. Проиллюстрирована реализация алгоритмов на предметно-ориентированном языке. Предложено проектное решение, позволяющее работать с абстракциями предметной области «вопрос-ответных» задач лингвистической направленности, не зависящее от области применения созданных компонентов. Однако их предполагаемое применение – область игрового электронного обучения. Следующая ступень – развитие темы каркаса и предметно-ориентированного языка – создание инструмента работы с языком. Речь идёт о так называемом визуальном предметно-ориентированном языке. Реализованные программно, инструменты такого языка позволяют создавать модели конкретных проблемных ситуаций и порождать из них готовый к использованию код. Применительно к тому, что было описано в статье, такой язык позволил бы создавать алгоритмы лингвистических задач (листинги 3 - 6) не вручную, а с помощью составления диаграмм заданной

нотации с последующей генерацией программного кода и помещением его в соответствующие места программы (точки расширения в листинге 2).

## Литература


1. Гусс С.В. Обучающие программы на основе игр лингвистической направленности // Информатика и образование. М. : ОАО «Московская газетная типография». 2009. № 11. С. 123-124.
2. Гусс С.В. Элементы повторного использования для программных систем учебного назначения. Концептуальное проектирование и анализ // Математические структуры и моделирование. Омск : ООО «УниПак». 2009. Вып. 20. С. 78-92.
3. Гусс С.В. Использование компьютерных лингвистических игр в процессе обучения // Открытое образование. М. : «CAPITALPRESS». 2010. № 1. С. 18-29.
4. Гусс С.В. Высокоуровневая модель семейства программных компонентов для поддержки занятий в игровой форме // Математические структуры и моделирование. Омск : «Омское книжное издательство». 2010. № 21. С. 44-54.
5. Гусс С.В. Модель каркаса программных компонентов поддержки занятий лингвистической направленности в игровой форме // Прикладная информатика. М. : ООО «Маркет ДС Корпорейшн». 2010. Вып. 3 (27). С. 62-77.
6. Гусс С.В. Проблема повторного использования в разработке семейства игровых программных систем учебного назначения // Вестник Омского университета. Омск : Издательство ОмГУ им. Ф.М. Достоевского. 2010. № 4. С. 147-149.
7. Гусс С.В. Пакет вспомогательных средств для построения семейства программных систем в определённой предметной области // Программная инженерия. М. : «Новые технологии». 2011. № 1. С. 25-33.
8. Furtado A., Santos A. Using Domain-Specific Modeling towards Computer Games Development Industrialization. URL: `http://www.dsmforum.org/events/DSM06/Papers/1-Furtado.pdf` (дата обращения: 6.04.2011).
9. Furtado A., Santos A., Ramalho G. A Computer Games Software Factory and Edutainment Platform for Microsoft .NET. URL: `http://www.cin.ufpe.br/~awbf/files/IET_SharpLudus.pdf` (дата обращения: 6.04.2011).
10. Furtado A., Santos A. Extending Visual Studio .NET as a Software Factory for Computer Games Development in the .NET Platform. URL: `http://www.cin.ufpe.br/~awbf/files/IVNET2006_SharpLudus.pdf` (дата обращения: 6.04.2011).
11. Furtado A., Santos A. Defining and Using Ontologies as Input for Game Software Factories. URL: `www.cin.ufpe.br/~awbf/files/SBGames2006_GameOntology.pdf` (дата обращения: 6.04.2011).
12. Куракин А.С. Опыт использования технологий дистанционного обучения в корпоративном образовательном проекте // Открытое образование. М. : «CAPITALPRESS». 2010. № 6. С. 57-68.
13. Шабалина О.А. Применение компьютерных игр для обучения разработке программного обеспечения // Открытое образование. М. : «CAPITALPRESS». 2010. № 6. С. 19-25.

14. Перевозчикова М.С. Создание компьютерной игры – эффективный инструмент самообразования // Информатика и образование. М. : ОАО «Московская газетная типография». 2009. № 11. С. 115-117.
15. Амблер С. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. СПб.: Питер, 2005. 412 с.
16. Арлоу Д., Нейштадт А. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. 2-е издание. СПб. : Символ-Плюс, 2007. 624 с.
17. Липаев В.В. Экономика производства программных продуктов. Издание второе. М. : СИНТЕГ, 2011. 352 с.
18. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. СПБ. : Питер, 2008. 366 с.
19. Гринфилд Д., Шорт К., Кук С., Кент С., Крупи Д. Фабрики разработки программ: потоковая сборка типовых приложений, моделирование, структуры и инструменты. М. : ООО «И.Д. Вильямс», 2007. 592 с.
20. Коплиен Д. Мультипарадигменное проектирование для C++. СПб. : Питер, 2005. 235 с.
21. Чарнецки К., Айзенекер У. Порождающее программирование: методы, инструменты, применение. СПб. : Питер, 2005. 731 с.
22. Эванс Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. М. : ООО «И.Д. Вильямс», 2011. 448 с.
23. Cook S., Jones G., Kent S. Domain-Specific Development with Visual Studio DSL Tools / United States of America : Addison Wesley Longman, Inc., 2007. 563 p.
24. Sutcliffe A. The Domain Theory: Patterns for Knowledge and Software Reuse. United States of America : CRC Press, 2002. 424 p.

*Математические структуры и моделирование*
2011, вып. 23, с. 76–85

УДК 004.414.38

# МЕТОДИКИ ПРИОРИТИЗАЦИИ ТРЕБОВАНИЙ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

**А.В. Непомнящих**

Рассматриваются методики приоритизации требований в проектах по разработке ПО и различные аспекты внедрения данных методик.

## Введение

По статистике два из трёх проектов терпят неудачу [4]. Таким образом, практически любой проект можно считать «безнадёжным» в той или иной степени.

По исследованиям 2006 года [5] одним из ключевых факторов успеха является правильная работа с требованиями.

Известный эксперт в области менеджмента программных проектов Эдвард Йордон [1] считает, что самое важное для успешного завершения проекта – это приоритизация требований:

> ...Если в этот момент вы решили, что у вас нет времени читать всю книгу, скажу только одно слово, которое может окупить время, потраченное на чтение предисловия: приоритетность (triage). Если вы участвуете в безнадёжном проекте, почти наверняка окажется недостаточно ресурсов, чтобы реализовать всю функциональность и возможность ПО, которые требуются конечному пользователю, в рамках утверждённого плана и бюджета. Так или иначе придётся решать, какие возможности следует реализовывать в первую очередь, а какими можно пожертвовать. Действительно, некоторые из незначительных возможностей не будут реализованы никогда, поэтому самое лучшее – это дать им спокойно умереть собственной смертью. Другие возможности являются достаточно важными, но также относительно легко реализуемыми, например, с помощью поставляемой библиотеки классов или используемых вами CASE-средств. Говоря языком медиков, эти возможности выживут сами по себе. Успех или неудача безнадёжного проекта зачастую зависит от способности проектной команды выделить критические функции и возможности,


Омский государственный университет им. Ф.М. Достоевского
E-mail: anepomnyaschih@gmail.com

> которые нельзя реализовать, не вкладывая значительные ресурсы и энергию...
>
> *Предисловие*

> ...Один из ключевых моментов в безнадёжных проектах состоит в том, что некоторые требования не просто останутся невыполненными до официального срока завершения проекта, но и не будут выполнены никогда. Если предположить, что известное правило «80-20» справедливо, то проектная команда сможет добиться 80 процентов «эффекта» от разработанной системы, реализовав 20 процентов требований к ней – при условии правильного выбора этих 20 процентов. И, поскольку пользователю, как правило, не терпится получить работающую систему задолго до того срока, который проектная команда считает разумным, то он может взять эти готовые 20 процентов, приступить к их использованию и навсегда позабыть об оставшихся 80 процентах функций системы...
>
> *Гл. 5.1 Концепция «triage»*

Другой эксперт, Джоэл Спольски также описывает этот факт [2]:

> Если бы мы не «отложили» все те «супер-важные» требования, разработка Excel 5 заняла бы вдвое больше времени и включала бы 50% бесполезной функциональности, которую бы требовалось сопровождать, для обратной совместимости, до конца времён.

В данной работе рассматриваются возможные проблемы при введении приоритизации в практику выполнения проектов по разработке ПО и предлагаются методы решения данных проблем.

## 1. Что предлагают эксперты?

Йордон:

> Все требования должны быть распределены по трём группам: «необходимо выполнить», «следует выполнить», «можно выполнить».

И проектная группа в первую очередь должна сконцентрировать усилия на требованиях первой группы, затем второй и затем, если останутся ресурсы, на третьей группе.

Те же принципы используются и в современных «гибких» методологиях разработки ПО:

Кент Бек [6]:

> ...XP подразумевает, что возможности с наивысшим приоритетом будут реализованы в первую очередь... В рамках XP заказчик должен определить наименьший допустимый набор возможностей, которыми должна обладать минимальная работоспособная версия программы, имеющая смысл с точки зрения решения бизнес-задач...

В методологии SCRUM есть понятие *очереди требований*. Таким образом, заказчик лишается возможности поставить каким-либо требованиям равный приоритет и убирается неявный конфликт, когда команду заставляют реализовывать все требования сразу.

В производственной системе Тойота также есть подобное понятие – *вытягивание* и *система канбан* [3]. Данные инструменты служат для той же цели: процесс-потребитель «вытягивает» работу у предыдущего процесса в цепи. Заказчик – у тестировщика, тестировщик – у разработчика, разработчик – у аналитика, аналитик вытягивает требования у заказчика. У процесса-потребителя есть только ограниченное количество карточек, по которым он может заказать продукт. Так как заказчик тоже является потребителем в данной цепи, то ему приходится выбирать, какие требования должны быть реализованы в первую очередь.

## 2. Определения

Требования бывают разных уровней. Требования верхнего уровня — это, фактически, бизнес-цели заказчика системы. По мере проработки аналитиками требования детализируются и уточняются. Проработанные требования нижнего уровня содержат описание отдельных частей системы и взаимосвязей между ними. Самыми сильными являются связи между требованиями верхнего уровня и их потомками.

Таким образом, имеется *дерево требований*.

*Запросом на изменение (ЗИ)* будем называть некий запрос в *системе управления проектом (СУП)*, в котором содержится описание функциональности, которую необходимо реализовать. В данном запросе также должно быть указано – исходя из какого набора требований нужна данная функциональность. То есть имеется прямая взаимосвязь между запросами на изменение и требованиями к системе.

Соответственно, бывают следующие типы ЗИ:

1. Новое требование — когда необходимо реализовать ещё одну ветвь дерева требований.

2. Улучшение — когда необходимо внести изменения в систему в соответствии с изменением некоторых требований.

3. Дефект — когда обнаружено несоответствие работы системы набору требований.

## 3. Препятствия внедрению приоритизации требований со стороны заказчика

Если рассмотреть природу появления требований, то этот процесс грубо можно описать так:

1. У заказчика появляется необходимость в решении некоторой реальной проблемы. Или идея, подразумевающая некоторую конкретную выгоду.

2. Пока заказчик ищет исполнителя, он привыкает к своей идее, и со временем она начинает обрастать всё новыми и новыми уточнениями и улучшениями. «Раз уж я в это ввязался, то почему бы не включить в проект ещё и вот это, и это?» — думает заказчик.

3. Идея обрастает совсем экстремальными и инновационными требованиями, которые до сих пор никем не были реализованы.

Если принять известное правило «80-20», то из вышесказанного можно сделать два вывода:

1. 80% выгоды находятся в 20% начальных требований (1).

2. 80% рисков содержатся в 20% поздних требований (3). Но эти требования, по сути, не решают изначальной задачи, а, в основном, «украшают» её.

Когда команда начинает просить заказчика выставить приоритеты, возникает следующая проблема: заказчику трудно отказываться от любых своих требований, особенно третьей группы (3), которые часто являются нестандартными, творческими, его собственными идеями — плодами его фантазии.

А как мы уже поняли, от них-то и следует отказаться в первую очередь, если возникнет проблема со сроками или ресурсами.

Во-вторых, заказчик может, в принципе, не понимать, как можно сортировать требования, ведь он представляет продукт цельным, и, например, уже согласована спецификация конечного продукта.

В такой ситуации нужно начинать переговоры и постараться объяснить заказчику необходимость сортировки требований:

1. Нужно объяснить, что требования поддаются сортировке (см. 3.1.).

2. Также хорошо помогает концепция очерёдности требований: «если бы вы могли получать требования последовательно, по очереди, в каком порядке вы бы хотели их получить? Какие требования помогли бы вам получить максимальную выгоду уже сейчас?»

3. Вместо слов «давайте уберём это требование» лучше использовать слова «давайте выполним его немного позже».

4. Команда и заказчик должны осознавать, что с любым проектом по разработке ПО всегда сопряжены огромные риски и доля неопределённости: по статистике [4] две третьих проектов по разработке ПО проваливаются. Таким образом, «если случится некоторая проблема, то мы можем что-то не успеть сделать к сроку. Чем бы вы могли пожертвовать в такой безвыходной ситуации? Да, мы полностью понесём ответственность по контракту, но по факту срок может быть сорван, и мы хотим, чтобы вы наименее от этого пострадали».

И даже если заказчик согласился разбить требования на три вышеупомянутых группы, он всё равно будет интуитивно пытаться всем требованиям поставить приоритет «необходимо». Даже если при старте проекта требования были упорядочены, то после нескольких первых итераций большинство требований снова получают высший приоритет – заказчику приходится каждую итерацию делать нелёгкий выбор, и он склоняется вообще его не делать.

Поэтому лучше всего зарекомендовал себя следующий подход: предоставляем заказчику список требований с их примерными оценками и просим упорядочить их по важности и желаемому порядку поступления. Не выставить числовые приоритеты, а именно упорядочить. И предупреждаем, что в этом порядке мы и будем над ними работать. То есть если возникнут непредвиденные трудности, то к сроку будет поставлено скорее то, что находится наверху списка. Таким образом, мы неявно заставляем заказчика упорядочить их линейно.

Когда заказчик начинает понимать, что он может регулировать ход проекта и менять порядок выполнения требований, он может начать использовать это неправильно и выставлять в начало требования третьей группы (см. раздел 3.), которые являются интересными для него, но не дают на самом деле большой выгоды.

В таком случае помогает:

1. Увеличение длительности итераций – у заказчика уже меньше средств для «игр» с требованиями. И если он хочет получить в результате итерации продукт, который можно продавать, то ему придётся включить некоторые требования первой группы (см. раздел 3.).

2. Постоянное отслеживание объёма оставшихся работ и оставшихся ресурсов и сроков. Эти показания передаются заказчику, и делаются рекомендации — можно ли ещё менять требования [7].

### 3.1. Независимость требований

Как мы уже поняли, приоритизация очень важна. Но как можно дать заказчику возможность менять очерёдность реализации требований, если все требования взаимосвязаны? Ведь порядок реализации требований будет сильно влиять на оценки сроков и стоимости реализации. А без оценок заказчик не может сортировать требования, т.к. сортировка выполняется как раз по соотношению выгода/стоимость для каждого требования.

1. Чтобы оценить соотношение выгода/стоимость, заказчику достаточно знать грубые оценки. Высокая точность оценок здесь не так важна, как при определении состава итерации, т.е. сроков поставки.

2. Если при старте проекта требования были упорядочены и выделены наиболее важные, то в дальнейшем данный порядок будет меняться незначительно. Более того, скорее всего часто будет меняться порядок только в верху списка, когда при старте очередной итерации заказчику нужно

будет выбрать, что в неё включить. Либо когда реализацию некоторого требования решили отложить.

3. Обычно требования в пределах одного слоя одной ветви дерева требований являются независимыми и порядок их реализации не важен. Важно только, когда данный слой в принципе начнёт разрабатываться (см. раздел 4.2.).

4. После выполнения каждого требования будет производиться переработка кода, которая позволит сохранить достаточно гибкую архитектуру проекта. Во-вторых, команда видит все будущие требования, даёт им оценки и, следовательно, представляет, куда движется проект. Таким образом, меньше вероятности столкнуться с ситуацией, когда очередное требование нельзя реализовать в рамках текущей архитектуры без её значительной переработки.

## 4. Препятствия внедрению приоритизации требований со стороны команды

### 4.1. Иерархическая структура работ

Заметим, что на практике дерево требований часто путают с иерархической структурой работ.

> *Иерархическая структура работ (ИСР)* – это ориентированная на результаты иерархическая декомпозиция работ, которые должна выполнить команда проекта для достижения целей проекта и создания требуемых результатов; на каждом более низком уровне ИСР представляет всё более детальное описание работ по проекту.
>
> *PMBOK, [8]*

Дело в том, что оценка стоимости проекта, согласно [8], выполняется именно по ИСР. Поэтому часто заказчик в описании стоимости проекта получает именно ИСР.

Здесь появляется ещё одна проблема: заказчик не может приоритизировать ИСР! Он может приоритизировать только требования! Поэтому оценки стоимости должны выставляться для требований. Как это сделать, описано в разделе 3.1.

### 4.2. Проблема первой итерации

Когда начинается разработка очередной ветви системы, слоя в этой ветви, то обычно необходимо подготовить некоторую инфраструктуру для данного слоя. Например, требование – это «рисование графических примитивов», а подтребования — «овал», «прямоугольник», «линия». Таким образом, в начале реализации необходимо будет подготовить возможность для рисования примитивов вообще, а затем уже разработать конкретные примитивы.

Такая необходимость возникает всегда, когда речь заходит о подобии. Программист стремится всё универсализировать, всю подобную функциональность выделить в отдельные классы и использовать их повторно.

Потому при старте проекта первая итерация, как правило, критична тем, что нужно создать функциональность, которая будет приносить выгоду бизнесу, но при это большую часть итерации приходится потратить не на создание полезного, а на создание того самого каркаса, инфраструктуры.

Предлагается два решения данной проблемы:

1. Использовать готовые каркасные решения. Фреймворки и CMS.

2. Использовать подход методологии eXtreme Programming (XP): "keep it simple". Не нужно делать того, что потом может не понадобиться: не стоит выделять каркас до того, как он пригодился. В каждой итерации делается переработка кода, на которую выделяется специально отведённое время.

Если вернуться к примеру, можно было бы поискать готовые компоненты рисования векторной графики. В таком случае мы берём на себя риск, что переделка компонента под наши конкретные нужды может занять много времени.

Если готовых компонентов не нашлось, нужно начать реализовывать решение для овала самостоятельно. Предположим, что про прямоугольник и линию заказчик ещё ничего не сказал. Тогда реализация рисования овала уместилась бы в одном классе. Затем нам сообщают про прямоугольник и линию. Вот тогда мы перерабатываем код, выделяем инфраструктуру рисования примитивов в абстрактные базовые классы и добавляем необходимые примитивы.

Казалось бы, быстрее было бы сразу заготовить необходимую инфраструктуру и её использовать. Но мы *не знали заранее*, понадобится ли она нам. А абстракции ухудшают читаемость и сопровождаемость кода. Возможно, мы бы потратили время на то, что потом будет не нужно, а поддерживать эту функциональность всё равно пришлось бы.

В данном же случае всего мы затратили времени больше, но получили большую выгоду — график добавления ценности в систему стал более гладким. Исчезла ситуация когда заказчик долгое время не получает полезной функциональности из-за того, что разработчики делают архитектуру чересчур гибкой, хотя большая часть этой гибкости, скорее всего, в будущем и не понадобится [9].

## 5. Проблемы менеджмента при внедрении приоритизации требований

В процессе исследования данной проблемы удалось выделить нижеследующие группы технических средств.

### 5.1. Багтрекеры

К багтрекерам относятся такие инструменты, как JIRA, Redmine, Bugzilla, Assembla.

Данные системы содержат в себе ЗИ различных типов (см. раздел 2.) в общей неупорядоченной базе. Хотя каждый ЗИ имеет различные атрибуты, в том числе и атрибут «приоритет», но очередь из них построить сложно.

Главная проблема данных систем — нет чёткой приоритизации. Команда и заказчик должны искусственно поддерживать актуальность приоритетов задач.

Зато данные системы сильно распространены, хорошо конфигурируются и могут поддерживать процесс самых различных проектных команд, будучи достаточно привычными и для заказчиков.

### 5.2. Системы поддержки «гибкой» разработки

К ним относятся: Pivotal Tracker, Agile Rally и др.

Содержат средства поддержки стандартных практик «гибкой» разработки. Например, строго ориентированы на итеративность. То есть все требования должны быть строго разнесены по итерациям, и нет возможности не использовать данную практику.

Главным положительным моментом для нас является то, что требования в данных системах представляют собой упорядоченный список – чем выше в списке, тем выше приоритет. Таким образом, заказчику и команде просто приходится сортировать требования и выделять важнейшие.

Отрицательные моменты:

1. Жёстко закреплённый процесс разработки. Низкие возможности конфигурирования. Проектной команде придётся подстраивать свой процесс под используемый инструмент.

2. Данные системы поддерживают достаточно небольшие команды (до 9 человек). Для большей команды просто не хватит возможностей пользовательского интерфейса данных систем. Также в данных системах практически отсутствуют средства документирования требований – можно указать лишь краткое описание задачи, что является проблемой для большой команды, где нет возможности часто встречаться всем её членам.

3. Мало распространены и потому непривычны большинству команд и заказчиков.

### 5.3. Требование в системе управления проектом

Таким образом, чтобы иметь возможность приоритизации, мы получаем достаточно жёсткое определение нового требования:

1. Каждое требование должно храниться отдельно.

2. У требования должен быть атрибут «приоритет» (см. 5.5.), либо требования должны храниться в линейно упорядоченном списке.

3. Требование должно быть понятным заказчику, чтобы он мог оценить выгоду от его реализации.

4. Требование должно иметь атрибуты – оценки затрат бюджета и времени.

5. Требование должно быть именно требованием, а не элементом ИСР.

Этих критериев достаточно для возможности сортировки требований по соотношению выгода/стоимость.

### 5.4. Дефекты

Дефект – это найденное несоответствие функциональности и требований. По сути, дефект – это незавершённая часть функциональности. Поэтому, когда найден дефект, должен быть переоткрыт соответствующий ЗИ с комментарием, описывающим дефект.

Но обычно найденные дефекты оформляются в системе отдельно, т.к. поиск дефектов — отдельная от самого программирования процедура, которая выполняется не разработчиками, а отдельными участниками проекта для повышения качества тестирования. Они могут не знать, какое конкретное требование привело к данному дефекту, потому оформляют его отдельно.

С приоритизацией дефектов проблем не возникает, кроме единственной: какой приоритет им ставить? Ведь заказчику хочется получить новую функциональность. Поэтому часто бывает, что дефекты получают достаточно низкие приоритеты, а ПО – соответствующее качество... Некоторые рекомендации по этому поводу есть в [10].

### 5.5. Глобальные и локальные приоритеты

В самых популярных системах управления проектами на сегодняшний день, таких как Atlassian JIRA, Microsoft Visual Studio Team Suite, т.е. Bugtrackers, отсутствует понятие очерёдности требований.

Данная проблема этих систем может быть решена следующим образом. Создаются две группы приоритетов — глобальные и локальные приоритеты.

Глобальные приоритеты выставляются при первичной сортировке требований. Данных приоритетов может быть сколько угодно – всё зависит от способности заказчика к упорядочению своих пожеланий. Минимально необходимое количество, в соответствии с рекомендацией Э. Йордона, это три приоритета: «необходимо», «важно», «желательно». Каждое требование проекта имеет некоторый глобальный приоритет.

При формировании списка требований на очередную итерацию берутся требования с наивысшим глобальным приоритетом, и им присваивается некоторый локальный приоритет. Минимальное необходимое количество локальных приоритетов: «обещанное» (Promised) и «желательное» (Stretched). Локальный

приоритет необходим как сигнал команде, какие из требований могут быть вычеркнуты в случае срабатывания рисков во время итерации. Сроки поставки сообщаются для всех требований, включённых в итерацию, но обговаривается, что «желательные» могут быть опущены в случае срабатывания рисков (см. 3.).

## 6. Рискованные требования — первыми

Таким образом, обоснована необходимость приоритизации и её осуществимость. Теперь нужно рассмотреть требования в контексте рисков. Дело в том, что с реализацией каждого требования сопряжены определённые риски, то есть потенциальные проблемы, которые могут осуществиться с определённой долей вероятности.

Если взглянуть на приоритизацию с точки зрения рисков, то проблема такова: если какие-то риски, сопряжённые с определёнными требованиями, реализуются на раннем этапе, то, возможно, будет выгоднее отменить проект, т.к. для реализации оставшихся требований не хватит бюджета или времени.

### Литература


1. Йордон Э. Путь камикадзе. М. : Лори, 2008. 289 с.
2. Spolsky J. Evidence Based Schedulling. URL: `http://www.joelonsoftware.com/items/2007/10/26.html` (дата обращения: 5.11.2010).
3. Лайкер Дж., Майер Д. Практика дао Toyota. Руководство по внедрению принципов менеджмента Toyota. М. : Альпина Паблишерз, 2009. 584 с.
4. Standish Group. CHAOS 2009. URL: `http://www.standishgroup.com/newsroom/chaos_2009.php` (дата обращения: 5.11.2010).
5. Standish Group. The Standigh Group Report. URL: `http://www.projectsmart.co.uk/docs/chaos_report.pdf` (дата обращения: 5.11.2010).
6. Бек К. Экстремальное программирование. СПб. : Питер, 2002. 224 с.
7. Дорофеев М. Статистика и Окно неопределенности. URL: `http://www.slideshare.net/Cartmendum/traininglabs09-part-3-of-4` (дата обращения: 5.11.2010).
8. Руководство к своду знаний по управлению проектами. Четвёртое издание // Project Management Institute. 2008. 241 с.
9. Поппендик М., Поппендик Т. Бережливое производство программного обеспечения. М. : Вильямс, 2010. 256 с.
10. Спольски Дж. Тест Джоэла: 12 шагов к лучшему коду. URL: `http://russian.joelonsoftware.com/Articles/TheJoelTest.html` (дата обращения: 5.11.2010).

*Математические структуры и моделирование*
2011, вып. 23, с. 86–98

УДК 004.4

# ПРОЕКТИРОВАНИЕ ПОШАГОВЫХ ОНЛАЙН-ИГР

**Е.В. Непомнящих**

Проводится анализ требований, предъявляемых к современным пошаговым онлайн-играм, и наиболее разумных путей их разработки. Приводится универсальный шаблон базы данных, протокола клиент-серверного взаимодействия и иерархии классов на стороне сервера.

## Введение

Современная индустрия информационных технологий (ИТ) основана на поиске и использовании универсальных шаблонов проектирования и разработки программных приложений. Данная статья является кратким обобщением наилучших практик, используемых при разработке пошаговых онлайн-игр, с кратким анализом и личным опытом автора.

Пошаговая игра — это жанр игр, основной особенностью которого является то, что игроки совершают ходы по очереди. К пошаговым играм относятся:

– пошаговые стратегии;

– карточные игры;

– настольные игры (шахматы, го, монополия и др.).

**Замечание 1.** Пошаговые игры накладывают меньше ограничений на сложность протокола взаимодействия по сравнению с играми в реальном времени. А именно, время реакции на то или иное событие, а значит, и качество сети ключевой роли не играют. Игроку обычно отводится от 10 секунд времени на принятие решения.

В большинстве пошаговых игр в любой момент времени решение принимает ровно один игрок. Следовательно, сужается множество запросов, на которые сервер должен адекватно реагировать.

Поэтому при создании протокола следует ориентироваться прежде всего на простоту его реализации и поддержки. Это позволит разработчику извлечь большую прибыль за меньшее время.


Омский государственный университет им. Ф.М. Достоевского
E-mail: enepomnyaschih@gmail.com

## 1. Предварительный анализ требований

Ниже перечислены требования, предъявляемые к большинству современных браузерных онлайн-игр:

1. Пользователь заходит на сайт и видит список доступных столов.
2. Пользователь может «сесть» за любой стол, если там есть свободные места.
3. При желании, пользователь может играть за несколькими столами одновременно.
4. Игра начинается, как только за столом не остаётся свободных мест.
5. Если пользователь встаёт из-за стола во время игры, его место занимает искусственный интеллект (не самый умный), пока пользователь не вернётся.
6. Время на раздумье ограничено (если долго думает, значит покинул игру).
7. Игра за любым столом ограничена по количеству ходов.

Такие требования наиболее хорошо подходят для игры на очки или деньги. Требования №1 – №4 представляют собой лучшие традиции пошаговых онлайн-игр. Требование №5 мотивирует игрока закончить игру, даже если он проигрывает. Вместе с требованиями №6 и №7 оно гарантирует, что всякая игра ограничена по времени. Пример игры, удовлетворяющей поставленным требованиям — KDice (`http://kdice.com`).

При проектировании базы данных и протокола клиент-серверного взаимодействия будем опираться на перечисленные требования.

### 1.1. Умный или глупый клиент?

Безусловно, сервер должен полностью владеть логикой приложения (правилами игры), чтобы предупредить его потенциальный взлом. Но стоит ли обучать бизнес-логике клиента?

Это напрямую зависит от того, какой размер имеет полный объем данных о состоянии игры. Если объем данных велик, долго собирается на сервере и передаётся клиенту, то имеет смысл часть логики реализовать на клиенте, чтобы разгрузить сервер. Например, в популярной игре Civilization IV датчик используемой памяти всегда зашкаливает. Возможно ли создать нечто подобное, оставив на клиенте исключительно пользовательский интерфейс?

С другой стороны, чем умнее клиент, тем дороже обойдётся разработка игры.

**Замечание 2.** От «эрудиции» клиента время разработки сервера никак не зависит. Если пользователь захочет перезагрузить окно браузера, то серверу придётся собирать и компоновать все данные об игре для передачи их клиенту.

Умный клиент может ускорить работу приложения, но он всегда потребует дополнительных ресурсов для своей разработки.

Следующий тест поможет сделать выбор:

**Позволяет ли объем канала?** Оценивается средний вес полного объема данных о состоянии игры. Далее, умножается на среднее количество запросов к серверу в секунду. Если полученное число превысит объем исходящего канала передачи данных, то глупый клиент недопустим. Если это число превысит 20% исходящего канала, то стоит призадуматься, потянет ли?

**Велика ли трудоёмкость?** Оценивается трудоёмкость алгоритма сбора данных об игре (в долях секунды). Учитываются все запросы к базе данных. Далее, умножается на среднее количество запросов серверу в секунду. Если время превысит одну секунду, то глупый клиент недопустим. Если это число превысит 200 мс, то стоит призадуматься, потянет ли?

Если объем канала позволяет и трудоёмкость сбора данных невелика, то рекомендуется использовать глупый клиент ввиду его невысокой стоимости.

### 1.2. Однонаправленный или двунаправленный протокол?

Самый обычный протокол HTTP-взаимодействия между клиентом и сервером (**первый вариант**) предполагает однонаправленные запросы от клиента к серверу. Клиент посылает запрос — сервер на него отвечает. Поскольку состояние игры периодически меняется по причине хода одного из игроков, запросы приходится посылать довольно часто с заданной периодичностью.

Тем не менее возможен и **второй вариант**: клиент и сервер посылают друг другу запросы обоюдно. Клиент посылает запрос, если пользователь кликнул мышкой. Сервер оповещает об этом остальных клиентов. Обычно клиент спит и ждёт сообщений от сервера.

**Замечание 3.** Запрос полного состояния игры все равно необходимо реализовать в том случае, если требуется позволять участнику продолжать игру даже после разрыва соединения.

Есть ещё и **третий вариант**. Все клиенты и сервер посылают запросы во всех направлениях. Этот вариант хорошо подошёл бы для игры в реальном времени, поскольку он очень быстрый. Но для реализации пошаговой игры он подходит не очень хорошо, потому что нас скорость работы протокола не сильно интересует, зато он самый сложный.

Следующий тест поможет выбрать путь реализации. Если все ответы «да», то следует использовать обоюдное общение клиента и сервера:

1. Ваша технология позволяет использовать двунаправленные запросы (например, в JavaScript на момент выпуска данной статьи это невозможно).

2. Вы владеете технологией достаточно хорошо (иначе затраты на изучение могут оказаться неоправданно велики).

3. У Вас есть запас времени для того, чтобы реализовать рассылку сообщений клиентам.

### 1.3. Структура базы данных

#### 1.3.1. Игроки (Players)

1. Идентификатор игрока (playerId) — первичный ключ
2. Имя игрока (playerName)
3. Информация «о себе»

#### 1.3.2. Сессии (Sessions)

1. Идентификатор сессии (ticket) — первичный ключ
2. Идентификатор игрока (playerId) — внешний ключ

#### 1.3.3. Столы (Tables)

1. Идентификатор стола (tableId) — первичный ключ
2. Данные о текущем состоянии игры

#### 1.3.4. Места за столами (Slots)

1. Идентификатор места (slotId) — первичный ключ
2. Идентификатор стола (tableId) — внешний ключ, альтернативный ключ
3. Идентификатор игрока (playerId) — внешний ключ, альтернативный ключ
4. Номер места (slotIndex)

#### 1.3.5. Игровые события (Events)

1. Идентификатор события (eventId) — первичный ключ
2. Идентификатор места (slotId) — внешний ключ
3. Порядковый номер события за столом (eventIndex)
4. Тип события (eventType)
5. Описание события (eventData) — произвольные сериализованные данные

## 2. Серверный интерфейс приложения (API)

### 2.1. Общие положения

В данной главе приводятся лишь те методы API, которые необходимы для реализации любой пошаговой игры, удовлетворяющей поставленным ранее требованиям.

Список методов:

1. **GetTables** — получение списка игровых столов.
2. **JoinTable** — подключение игрока к столу.
3. **GetTable** — получение полной информации о столе, включая текущее состояние игры.
4. **PlayerTurn** — набор запросов о действиях игрока.
5. **WakeUp** — вернуть управление игроку, если его место занял искусственный интеллект.

Кроме того, понадобится реализовать один из наборов:

| | Умный клиент | Глупый клиент |
|---|---|---|
| Односторонние запросы | GetEvents | GetUpdate |
| Двусторонние запросы | NotifyEvents | NotifyUpdate |

,

где:

1. **GetEvents** — получение списка произошедших событий
2. **GetUpdate** — получение обновленного состояния игры
3. **NotifyEvents** — оповещение о произошедших событиях
4. **NotifyUpdate** — оповещение об обновлении состояния игры

Большинство запросов требует авторизации (в заголовках `Cookie` должен быть прописан действующий идентификатор сессии). Если требуется дать возможность участникам играть «на нескольких досках одновременно», то почти все запросы должны принимать обязательный параметр `tableId`. Иначе игра должна определяться по идентификатору сессии игрока, что не позволит участникам подключаться к нескольким играм одновременно.

Если клиент умный, т.е. он владеет игровой бизнес-логикой, то списка событий ему достаточно, чтобы самостоятельно построить новое состояние игры. Если клиент глупый, то ему необходимо передавать полную информацию о текущем состоянии игры, а также список возможных действий, которые игрок может выполнить на данном этапе игры. Сервер без труда может передавать полную информацию, но объем передаваемых данных из-за этого несколько увеличивается. Даже если клиент глупый, ему все равно нужен список событий, чтобы показать их на экране со всей соответствующей анимацией.

### 2.2. Общий формат всех запросов

Некоторые веб-технологии (например, Adobe Flash на момент выпуска данной статьи) не дают возможность читать ответы с HTTP-кодом, отличным от `200 OK`. Поэтому все запросы должны возвращать ответы именно с этим HTTP-кодом.

В связи с этим роль HTTP-кода должно выполнять содержимое тела ответа. Например, можно использовать следующий универсальный формат ответа (JSON-версия):

```
{
    success : true или false,
    error   : null или { // пример ошибки
        code    : "ERROR_AUTHORIZATION",
        message : "You must login to call this action"
    },
    result  : произвольные данные
}
```

### 2.3. GetTables — получение списка столов

`GET /application/tables`, публичный доступ.
Возвращает список доступных столов для подключения.
По желанию, можно передавать параметры фильтрации, сортировки и пр.
Ответ:

```
[
    {
        tableId   : 10,
        tableName : "My favorite table",
        ... // другие краткие данные
    },
    ...
]
```

Возможные ошибки: нет

### 2.4. JoinTable — подключение игрока к столу

`GET /table/join`, авторизованный доступ.
Подключает игрока к столу.
Параметры:

1. `tableId` — обязательный целочисленный параметр, идентификатор стола

Ответ: `null`
Возможные ошибки:

1. `ERROR_AUTHORIZATION` — требуется авторизация
2. `ERROR_PARAM_INVALID` — обязательный параметр не указан или параметр принимает неправильное значение
3. `ERROR_NO_SLOT` — все места заняты
4. `ERROR_JOINED_ALREADY` — игрок уже сидит за этим столом

### 2.5. GetTable — получение полной информации о столе

`GET /table/info`, авторизованный доступ.
Возвращает полную информацию о столе, включая текущее состояние игры.
Параметры:

1. `tableId` — обязательный целочисленный параметр, идентификатор стола

Ответ:

```
{
    gameInfo: {
        players: [
            {
                playerId   : Int,
                playerName : String,
                ...
            },
            ...
        ],
        ...
    },
    ... // см. GetUpdate
}
```

Возможные ошибки:

1. `ERROR_AUTHORIZATION` — требуется авторизация
2. `ERROR_NOT_JOINED` — игрок не сидит за столом
3. `ERROR_PARAM_INVALID` — обязательный параметр не указан или параметр принимает неправильное значение

### 2.6. PlayerTurn — набор запросов о действиях игрока

`GET /table/turn, /table/pass, /table/double` и др., авторизованный доступ.
Выполняет ход или другое действие игрока.
Параметры:

1. `tableId` — обязательный целочисленный параметр, идентификатор стола
2. Другие параметры, зависящие от типа действия. Например, идентификатор карты в карточной игре

Ответ: `null` или данные о произошедшем событии (запись из таблицы Events).
Возможные ошибки:

1. `ERROR_AUTHORIZATION` — требуется авторизация
2. `ERROR_NOT_JOINED` — игрок не сидит за столом
3. `ERROR_PARAM_INVALID` — обязательный параметр не указан или параметр принимает неправильное значение
4. `ERROR_GAME_STATE` — данное действие запрещено на данном этапе игры. Возможно, игра уже закончилась

5. ERROR_CURRENT_PLAYER — сейчас ходит другой игрок
6. Другие ошибки, зависящие от действия. Например, нельзя дважды удваивать ставку

### 2.7. WakeUp — вернуть управление от искусственного интеллекта

`GET /table/wakeup`, авторизованный доступ.
Возвращает управление игроку, если его место занимает искусственный интеллект.
Параметры:
1. `tableId` — обязательный целочисленный параметр, идентификатор стола

Ответ: `null`
Возможные ошибки:
1. ERROR_AUTHORIZATION — требуется авторизация
2. ERROR_NOT_JOINED — игрок не сидит за столом
3. ERROR_PARAM_INVALID — обязательный параметр не указан или параметр принимает неправильное значение
4. ERROR_NOT_SLEEPING — искусственный интеллект не включён

### 2.8. GetEvents — получение списка произошедших событий

`GET /table/events`, авторизованный доступ.
Возвращает список произошедших событий, начиная с указанного порядкового номера.
Параметры:
1. `tableId` — обязательный целочисленный параметр, идентификатор стола
2. `fromIndex` — необязательный целочисленный параметр, номер события, начиная с которого вернуть события

Ответ:

```
[
    {
        eventIndex  : Int,
        eventType   : String,
        tableSlotId : Int,
        ... // другие данные
    },
    ...
]
```

Возможные ошибки:
1. ERROR_AUTHORIZATION — требуется авторизация
2. ERROR_NOT_JOINED — игрок не сидит за столом
3. ERROR_PARAM_INVALID — обязательный параметр не указан или параметр принимает неправильное значение

### 2.9. GetUpdate — получение обновлённого состояния игры

`GET /table/update`, авторизованный доступ.
Возвращает текущее состояние игры, список доступных действий игрока и список произошедших событий, начиная с указанного порядкового номера.
Параметры:

1. `tableId` — обязательный целочисленный параметр, идентификатор стола
2. `fromIndex` — необязательный целочисленный параметр, номер события, начиная с которого вернуть события

Ответ:

```
{
    gameState: {
        stateId: String, // текущий этап игры
        currentTableSlotId: Int, // текущий игрок
        ... // другие данные
    },

    gameDecision: {
        sleep: true/false, // включен ли иск. интеллект
        ... // данные о допустимых действиях игрока
    },

    gameEvents: ... // см. GetEvents
}
```

Возможные ошибки:

1. `ERROR_AUTHORIZATION` — требуется авторизация
2. `ERROR_NOT_JOINED` — игрок не сидит за столом
3. `ERROR_PARAM_INVALID` — обязательный параметр не указан или параметр принимает неправильное значение

### 2.10. NotifyEvents — оповещение о произошедших событиях

POST-запрос осуществляется от сервера к клиенту.
Оповещает о произошедшем событии.
Параметры равны свойствам произошедшего события (см. GetEvents).
Ответ: `null`

### 2.11. NotifyUpdate — оповещение об обновлении состояния игры

POST-запрос осуществляется от сервера к клиенту.
Передает новое состояние игры.
Параметры равны данным о новом состоянии игры (см. GetUpdate).
Ответ: `null`

## 3. Методика реализации серверной части

В данной главе пойдёт речь о том, как лучше построить процесс разработки для того, чтобы малыми усилиями, но достаточно качественно реализовать приложение пошаговой онлайн-игры. В параграфах описываются последовательные шаги, с которых следует начинать разработку. Следуя этим шагам, разработчику удастся в кратчайшие сроки подготовить предварительную версию сервера приложения для интеграции с клиентской частью.

### 3.1. Выделить этапы игры

Прежде всего следует определить, на какие этапы можно разбить игровой процесс. Этап игры — это некоторое состояние приложения, определяющее алгоритмы обработки запросов для конкретного игрового стола (`tableId`). Выделение этапов игры позволит разработчику без труда построить корректную и удобную иерархию классов на стороне сервера. Пример для популярной игры «Червы»:

1. Ожидание игроков
2. Сброс трех карт
3. Розыгрыш всех карт
4. Подведение итогов партии и возможный переход к следующей партии
5. Подведение итогов игры, распределение выигрыша

В результате должен получиться детерминированный автомат переходов между этапами игры. Переходы между ячейками автомата осуществляются при действиях игроков и имеют четкие условия. При проектировании сервера следует в первую очередь нарисовать этот автомат. Чем больше ячеек, тем прозрачнее становится принцип работы автомата, и тем проще становятся условия перехода между ячейками.

В приведённом примере имеются следующие переходы: 1 — 2, 2 — 3, 3 — 4, 4 — 2, 4 — 5. Начальное состояние — 1, конечное состояние — 5.

### 3.2. Создать классы этапов игры

Далее, необходимо создать иерархию классов, однозначно соответствующих этапам игры. Следует повсеместно опираться на паттерн «Шаблонный метод (Template method)» [5] для придания гибкости. Интерфейс базового класса получается примерно такой (используется псевдоязык):

```
class StageBase
{
    // Виртуальные абстрактные public-методы
    function getStageId();          // Идентификатор фазы игры

    // Фиксированные шаблонные public-методы.
    // Несут общий функционал
    function start();               // Когда автомат перешел в это состояние
    function daemon();              // "Демон", вызывается в начале запроса
```

```
    function joinTableAction();     // Обработчик /table/jointable
    function getTableAction();      // Обработчик /table/gettable
    function turnAction(cardId);    // Обработчик /table/turn
    function passAction();          // Обработчик /table/pass
    function doubleAction();        // Обработчик /table/double
    function wakeUpAction();        // Обработчик /table/wakeup
    function getEventsAction();     // Обработчик /table/getevents
    function getUpdateAction();     // Обработчик /table/getupdate
    ...

    // Виртуальные абстрактные protected-методы.
    // Эти методы вызываются из соответствующих public-методов
    function onStart();             // Перегружаемый вход в состояние
    function onDaemon();            // Перегружаемый "демон"

    // замечание: следующие методы по умолчанию выбрасывают
    // ошибку ERROR_GAME_STATE (запрещено на данном этапе игры)
    // Это ядро системы: то, что заставляет программу работать
    function onJoinTableAction();   // Перегружаемый /table/jointable
    function onGetTableAction();    // Перегружаемый /table/gettable
    function onTurnAction(cardId);  // Перегружаемый /table/turn
    function onPassAction();        // Перегружаемый /table/pass
    function onDoubleAction();      // Перегружаемый /table/double
    function onWakeUpAction();      // Перегружаемый /table/wakeup
    function onGetEventsAction();   // Перегружаемый /table/getevents
    function onGetUpdateAction();   // Перегружаемый /table/getupdate

    // Вспомогательные фиксированные шаблонные protected-методы
    function nextPlayer();          // К следующему игроку
    function afterRound();          // Закончился круг
    function runAI();               // Запуск ИИ
    function changeStage(next);     // Сменить этап игры

    // Вспомогательные виртуальные абстрактные protected-методы
    function onNextPlayer();        // Перегружаемая часть nextPlayer
    function onAfterRound();        // Перегружаемая часть afterRound
    function onRunAI();             // Перегружаемая часть runAI
    function onChangeState(next);   // Перегружаемая часть chartStage

    // Виртуальные protected-методы для сериализации состояния игры
    function getGameInfo();
    function getGameState();
    function getGameEvents(lastEvent);
    function getCurrentPlayer();
    function getPlayerCards();
    ...

    // Фиксированные protected-методы валидации запроса.
    // Их удобно вызывать в начале методов onXxxxxAction
    function requireJoin();             // ERROR_NOT_JOINED?
    function requireCurrentPlayer();    // ERROR_CURRENT_PLAYER?
    ...
}
```

Тем самым вносится максимально возможная полиморфность между этапами игры и, с другой стороны, уже реализуется общая часть функционала игры. Это лишь пример того, как это может выглядеть. Конечно, разработчик имеет полную свободу по внесению изменений в данный базовый класс.

Все остальные конкретные классы этапов игры пока остаются пустыми.

Также разработчику понадобится класс `GameModel`, являющийся адаптером над базой данных. Он предоставляет методы доступа к данным, различные игровые алгоритмы и все, что связано с правилами игры. Его тоже следует пока оставить пустым.

Шаблон обработчиков методов API приложения получается примерно такой:

1. Проверятся авторизация, параметры и другие условия
2. Блокируется стол по идентификатору `tableId` (произвольными объектами синхронизации), чтобы другие запросы не могли испортить состояния базы данных
3. Конструируется экземпляр `GameModel`
4. Конструируется экземпляр одного из потомков `StageBase`, в зависимости от текущего этапа игры
5. Вызывается метод `stage.daemon`
6. Вызывается метод `stage.xxxxxAction`, в зависимости от вызванного метода API
7. Стол разблокируется (PHP, например, это делает автоматически)

Вся дальнейшая разработка сводится к наполнению классов этапов игры и игровой модели. Об этом в следующем параграфе.

### 3.3. Полезные практики при программировании функционала игры

Некоторые практики программирования оказываются крайне полезными при разработке веб-приложений. Они проверены временем, и о них много пишут. Вот краткий список основных практик:

1. Функционально-ориентированная разработка (Feature Driven Development) [8]
2. Разработка через тестирование (Test Driven Development) [9]
3. Рефакторинг (Refactoring) [6]

Применяя методику последовательной (функционально-ориентированной) разработки к пошаговой игре, следует действовать в следующем порядке:

1. Создать интерфейс базового класса `StageBase` и унаследовать от него пустые классы всех этапов игры
2. Добавить проваливающиеся тесты:
    1. Что в игру может зайти 1, 2, 3, 4 игрока
    2. Что в игру не могут зайти неавторизованные пользователи
    3. Что в игру не может зайти больше 4-х игроков
    4. Что один и тот же игрок не может войти в одну игру дважды
5. Внести изменения, необходимые для того, чтобы данные тесты прошли

6. Добавить проваливающиеся тесты:
    1. Что при входе 4-го игрока этап игры меняется
    2. Что в новом состоянии игроки входить в игру не могут
3. Внести изменения, необходимые для того, чтобы данные тесты прошли
4. ... так далее

Процесс естественным образом укладывается в методику разработки через тестирование. Как только на очередном этапе разработки начнут проваливаться старые тесты, то стоит задуматься, какие действия в базовом классе следует разместить в других местах, что нужно добавить/удалить и пр. То есть преобразовать базовый класс путём рефакторинга с целью поиска ошибок.

## Заключение

Описанный в статье шаблон проектирования успешно применялся на предприятии в разработке ряда проектов.

### Литература


1. Soren Johnson. Game Developer Column 8: Turn-Based vs. Real-Time. URL: `http://www.designer-notes.com/?p=151` (дата обращения: 10.04.2011).
2. Amit's Game Programming Information. URL: `http://www-cs-students.stanford.edu/~amitp/gameprog.html` (дата обращения: 10.04.2011).
3. Материал из свободной энциклопедии. Пошаговая стратегия. URL: `http://ru.wikipedia.org/wiki/Пошаговая_стратегия` (дата обращения: 10.04.2011).
4. Зеленков Ю.А. Введение в базы данных. Ярославль : ЯрГУ, 1997. URL: `http://www.mstu.edu.ru/study/materials/zelenkov/ch_7_1.html` (дата обращения: 10.04.2011).
5. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. СПб. : Питер, 2008. 366 с.
6. Фаулер, М. Рефакторинг // СПб. : Символ-Плюс, 2003. 432 с.
7. Hall J.R. Programming Linux Games // San Francisco : Loki Software, Inc, 2001. 415 с.
8. Палмер С.Р., Фелсинг Д.М. Практическое руководство по функционально-ориентированной разработке ПО // М. : Вильямс. 304 с.
9. TDD - Разработка через тестирование (Test Driven Development). URL: `http://wiki.agiledev.ru/doku.php?id=tdd`

*Математические структуры и моделирование*
2011, вып. 23, с. 99–104

УДК 004.056.53

# АЛГОРИТМ ПОИСКА МОСТОВ ТИПА $\overrightarrow{t^*}$ И $\overleftarrow{t^*}$ В ГРАФЕ ДОСТУПОВ ДЛЯ ДИСКРЕЦИОННОЙ МОДЕЛИ БЕЗОПАСНОСТИ TAKE-GRANT

**Д.М. Бречка**

В статье приводится алгоритм поиска мостов типа $\overrightarrow{t^*}$ и $\overleftarrow{t^*}$ в графе доступов для дискреционной модели безопасности Take-Grant, основанный на классическом алгоритме поиска в ширину.

## Введение

Модель Take-Grant является одной из первых и наиболее глубоко проработанных в математическом плане дискреционных моделей безопасности [1–3]. Одним из наиболее существенных достижений классической модели Take-Grant можно считать выработанные условия, при соблюдении которых компьютерная система считается защищённой. Эти условия можно найти в двух теоремах об истинности предиката *«возможен доступ»*, сформулированных и доказанных для модели Take-Grant [2, 3]. В классической модели способ проверки выполнимости этих условий никак не оговаривается. Наиболее простым способом проверки, в концептуальном плане, можно считать полный перебор вершин и дуг графа доступов. Однако, как известно, алгоритм перебора имеет экспоненциальную вычислительную сложность, это означает, что выполняться такой алгоритм на обычной ЭВМ будет слишком медленно.

В более поздних работах, посвящённых модели Take-Grant, предлагаются альтернативные способы проверки защищённости компьютерной системы, не основывающиеся на предложенных в классической модели условиях. В работах [4, 5] предлагаются алгоритмы, имеющие полиномиальную сложность $O(N^3)$.

Данная статья является продолжением работ [6, 7], в которых описываются способы проверки безопасности компьютерной системы, основанные на тех условиях, что сформулированы в классической модели Take-Grant. В статье описывается полиномиальный алгоритм для поиска мостов типа $\overrightarrow{t^*}$ и $\overleftarrow{t^*}$ в графе доступов. Предлагаемый алгоритм основан на классическом алгоритме поиска в ширину [8].


Омский государственный университет им. Ф.М. Достоевского
E-mail: dbrechkawork@yandex.ru

## 1. Начальные условия и обозначения

Исходя из условий, сформулированных в модели Take-Grant, для того чтобы предикат *«возможен доступ»* был истинен в произвольном графе, необходимо, чтобы в этом графе были известны острова (tg-связные подграфы из вершин-субъектов), мосты, начальные и конечные пролёты мостов (пути в графе заданного вида). В данной статье ограничимся поиском мостов типа $\overrightarrow{t^*}$ и $\overleftarrow{t^*}$ — путей в графе, каждая дуга которых содержит метку $\overrightarrow{t}$ или $\overleftarrow{t}$ соответственно.

Пусть в графе уже известны острова, способ их нахождения приводится в [6, 7]. Нам нужно найти мост между двумя островами $I_1$ и $I_2$. Введём ряд обозначений. Будем обозначать дугу, содержащую метку $\overrightarrow{t}$, между вершинами графа $e_i$ и $e_j$ через $\overrightarrow{t}(e_i, e_j)$. Начальную вершину моста будем обозначать через $s$, а конечную — через $f$. Множество всех вершин-объектов исходного графа обозначим через $O$.

## 2. Мост $\overrightarrow{t^*}$

Рассмотрим поиск моста типа $\overrightarrow{t^*}$. Для начала опишем алгоритм неформально. В начале работы алгоритм разбивает все множество вершин-объектов исходного графа на два подмножества $O_r$ и $O_i$. В множество $O_r$ попадают те вершины, до которых существует мост заданного вида, все остальные вершины заносятся в множество $O_i$. В самом начале работы алгоритма в множестве $O_r$ находится только одна вершина — $s$ (начальная вершина). Далее алгоритм просматривает все дуги графа, инцидентные вершинам из множества $O_r$, и если обнаруживается дуга вида $\overrightarrow{t}$, соединяющая вершину $e_r$ из $O_r$ с вершиной $e_i$ из $O_i$, то $e_i$ удаляется из $O_i$ и заносится в $O_r$. После просмотра всех дуг графа в $O_r$ может быть занесено сразу несколько вершин, то есть мощность $O_r$ увеличится на некоторое число, а мощность $O_i$ уменьшится на это же число. Если вершина $f$ попадает в множество $O_r$, то это значит, что в графе существует путь заданного вида и алгоритм заканчивает работу. Иначе, процедура повторяется для измеренных $O_r$ и $O_i$. Однако возможна ситуация, когда после просмотра всех дуг в $O_r$ не будет добавлено ни одной вершины. Это возможно, когда между вершинами из $O_r$ и вершинами из $O_i$ не существует дуг вида $\overrightarrow{t}$. Чтобы отловить данную ситуацию, нужно проверить мощности множеств, которыми мы оперируем: если мощности множеств не изменились, значит, необходимо закончить работу алгоритма с выдачей сообщения о том, что моста, заданного между рассматриваемыми островами, не существует.

Формально алгоритм будет состоять из трёх основных этапов. Перед началом работы алгоритма разобьём множество $O$ на два подмножества $O_r$ и $O_i$, причём $O = O_r \cup O_i$.

**Этап 1.** Вершина $s$ заносится в множество $O_r$, все остальные вершины заносятся в $O_i$.

**Этап 2.** Просматриваются все дуги графа, началом которых являются вершины из $O_r$, если существует $\overrightarrow{t}(e_r, e_i)$, то $e_i$ заносится в множество $O_r$ и

удаляется из $O_i$. Здесь $e_r \in O_r$, $e_i \in O_i$. Когда все дуги, инцидентные вершинам из $O_r$ множества, будут просмотрены, переходим на третий этап.

**Этап 3.** Если после выполнения этапа 2 вершина $f$ оказалась во множестве $O_r$, то алгоритм заканчивает свою работу: мост заданного вида в графе существует. Если после выполнения этапа 2 мощности множеств $O_r$ и $O_i$ не изменились, алгоритм также заканчивает работу: моста заданного вида в графе не существует. В противном случае — возвращаемся на этап 2.

**Замечание 1.** В зависимости от задачи можно предусмотреть разные реализации алгоритма. Например, если требуется лишь показать наличие или отсутствие моста между указанными вершинами, то приведённого выше описания будет достаточно — алгоритм выдаёт сообщение о результатах своей работы. Если же требуется найти сам мост, то необходимо дополнительно каким-либо образом поддерживать множество отмеченных вершин и дуг. Например, после каждого шага второго этапа строить граф путей (который не обязательно будет деревом) или определённым образом окрашивать выбранные вершины и дуги.

Проведём оценку работы предложенного алгоритма. Пусть исходный граф содержит $N$ вершин. Так как граф ориентирован, то в случае его полносвязности количество дуг в нем будет равно $N(N-1)$. Количество повторений этапа 2 можно ограничить количеством вершин графа, так как в случае если после каждого выполнения этапа в множество $O_r$ будет заноситься по одной вершине, то мост в графе будет найден за $N$ шагов. В общем случае в множество $O_r$ за каждое выполнение этапа 2 будет заноситься более одной вершины, то есть мост будет найден меньше чем за $N$ шагов. Если же моста не существует, то на каком-то шаге не найдётся дуги нужного вида и алгоритм ничего не занесёт в множество $O_r$, то есть мощность множества останется неизменной, тогда алгоритм прервётся с выдачей соответствующего сообщения. Таким образом, сложность работы алгоритма можно оценить как или $O(N^3)$.

**Теорема 1.** *Приведённый выше алгоритм корректно находит мост типа* $\overrightarrow{t^*}$.

**Доказательство.** Во-первых нужно доказать, что алгоритм вообще закончит свою работу (сходимость алгоритма), во-вторых — что алгоритм найдёт мост нужного вида (корректность алгоритма).

Сходимость можно показать исходя из того, что множество вершин-объектов исходного графа конечно. Алгоритм разбивает множество $O$ на $O_i$ и $O_r$, причём $O = O_r \cup O_i$. На каждом шаге второго этапа алгоритма либо какая-то вершина удаляется из $O_i$ и заносится в $O_r$, при этом мощности обоих множеств соответственно изменяются, либо перемещения вершин не происходит — мощности множеств не меняются, и алгоритм заканчивает работу.

Пусть $|O| = M$. Возможно три варианта развития событий.

1. Все вершины из $O_i$ будут перенесены в $O_r$, тогда на следующем шаге алгоритм закончит свою работу, так как мощности множеств менять будет уже невозможно. Всего алгоритм совершит $M + 1$ шагов.

2. На шаге $k < M$ алгоритм обнаружит мост. Алгоритм закончит работу, при этом совершит $k$ шагов.

3. На шаге $k < M$ алгоритм обнаружит, что мощности множеств $O_i$ и $O_r$ не изменились. Алгоритм закончит работу, при этом совершит $k$ шагов.

Таким образом, алгоритм в любом случае закончит работу независимо от того, присутствуют в графе мосты или нет.

Корректность алгоритма покажем индукцией по длине моста ($n$). В качестве базы индукции мы не можем выбрать $n = 1$, так как это бы означало, что начальная и конечная вершина моста соединены дугой $\vec{t}$ и, следовательно, принадлежат одному острову. Поэтому в качестве базы выберем $n = 2$ и покажем, что алгоритм найдёт такой мост.

Ситуация, когда в графе присутствует мост длины 2, изображена на рисунке 1. С вершиной $s$ связана как минимум одна дуга вида $\vec{t}$, соединяющая вершины $s$ и $x$. С вершиной $x$, в свою очередь, связана как минимум одна дуга вида $\vec{t}$, соединяющая $x$ и $f$.

Вершина $s$ заносится в множество $O_r$ на первом этапе алгоритма. На втором этапе, при проверке всех дуг вида $\vec{t}$, инцидентных вершинам из множества $O_r$, будет обнаружена вершина $x$. Эта вершина будет занесена в $O_r$ и удалена из $O_i$. Однако мост еще не будет найден. Он обнаружится только после повторного выполнения второго этапа, когда вновь будут просматриваться дуги вида $\vec{t}$, инцидентные вершинам из множества $O_r$, т.е. будет обнаружена дуга, связывающая $x$ и $f$.

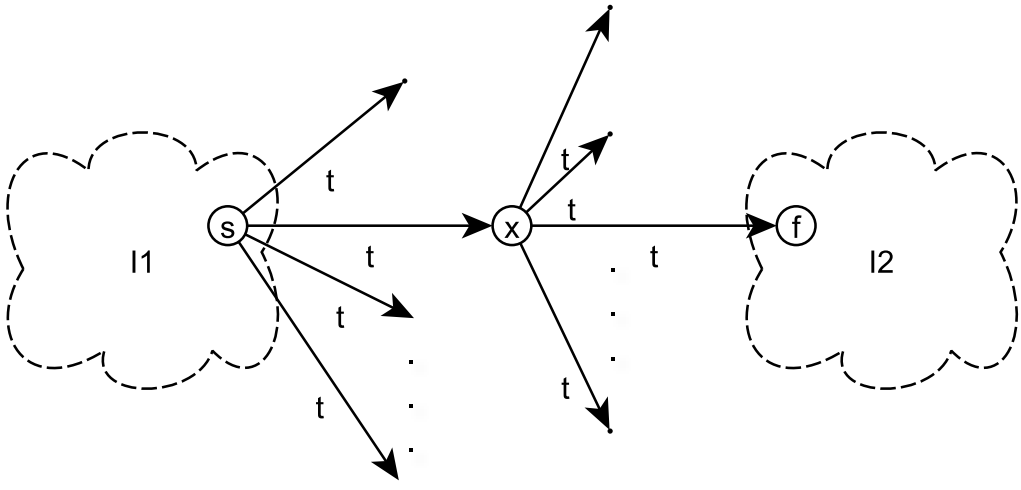


Рис. 1. Мост $\vec{t}^*$ длины 2

В качестве предположения индукции выберем утверждение, что для длины моста $n < k$, где $k > 2$ алгоритм находит мост правильно.

Шаг индукции: пусть длина моста равна $k$, алгоритм выполнил $k-2$й шаг, и на этом шаге в множество $O_r$ были занесены вершины $x_1, x_2, \ldots x_m$ (рис. 2). Согласно предположению индукции, до каждой из вершин $x_j$ мост найден правильно. Так как длина моста равна $k$, это означает, что как минимум между одной из $x_j$ и $f$ существуют дуги $\overrightarrow{t}(x_j, y)$ и $\overrightarrow{t}(y, f)$, являющиеся последними дугами искомого моста.

Применим алгоритм для каждой из $x_j$, как показано для базы индукции, алгоритм способен правильно найти мост, состоящий их двух дуг.

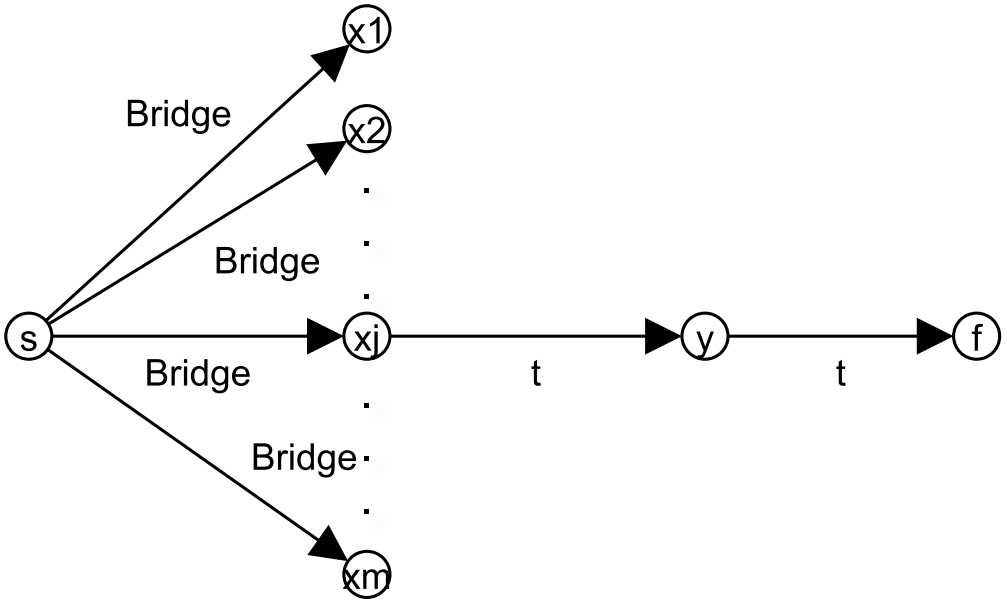


Рис. 2. Шаг индукции

■

## 3. Мост $\overleftarrow{t^*}$

Для моста типа $\overleftarrow{t^*}$ очевидно можно использовать описанный выше алгоритм, если на втором этапе вместо дуг типа $\overrightarrow{t^*}$ искать дуги типа $\overleftarrow{t^*}$. При этом все сказанное выше будет справедливо и для моста $\overleftarrow{t^*}$, в том числе трудоёмкость алгоритма также будет оцениваться как $O(N^3)$.

## Заключение

Возможность проверки предиката *«возможен доступ»* на истинность позволяет установить наличие или отсутствие каналов утечки информации между двумя выделенными субъектами компьютерной системы. Для того чтобы

совершить проверку истинности данного предиката, необходимо также предусмотреть способы поиска мостов типа $\overrightarrow{t^*}\overrightarrow{g}\overleftarrow{t^*}$ и $\overrightarrow{t^*}\overleftarrow{g}\overleftarrow{t^*}$, а также начального и конечного пролётов моста. В данной статье способы поиска этих структур не рассматриваются. Однако разработка полиномиальных алгоритмов поиска указанных структур может лечь в основу программных средств анализа безопасности информационных систем с целью выявления каналов утечки информации.

## Литература


1. Lipton R.J., Snayder L. A linear time algorithm for deciding subject security // Journal of ACM (Addison-Wesley). N.3. 1977. P.455-464
2. Теоретические основы компьютерной безопасности: Учебное пособие для вузов / Девянин П.Н. и др. М. : Радио и связь, 2001. 192 с.
3. Гайдамакин Н.А. Разграничение доступа к информации в компьютерных системах. Е. : Издательство Уральского Университета, 2003. 328 с.
4. Frank J., Bishop M. Extending the Take-Grant protection system. Technical report. Department of Computer science, University of California in Devis, 1996. 14 p.
5. Bishop M. Theft of information in the Take-Grant protection model // Computer security 3 (4). 1994. P.283–309.
6. Бречка Д.М. Алгоритмы анализа безопасности состояний компьютерной системы для модели Take-Grant // Математические структуры и моделирование. 2009. №20. С.160-172.
7. Бречка Д.М. Алгоритмы проверки безопасности состояний компьютерной системы в модели Take-Grant // Проблемы обработки и защиты информации. Книга 1. Модели политик безопасности компьютерных систем. Коллективная монография / Под общей редакцией С.В. Белима. Омск : ООО «Полиграфический центр КАН», 2010.
8. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М. : МЦНМО, 2000. 960 с.

*Математические структуры и моделирование*
2011, вып. 23, с. 105–107

УДК 004.056.5

# ОСОБЕННОСТИ РЕАЛИЗАЦИИ ПРОТОКОЛА ВЫРАБОТКИ ОБЩЕГО КЛЮЧА С ИСПОЛЬЗОВАНИЕМ ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ

**А.И. Журавлёв, Д.Н. Лавров**

В статье описываются особенности реализации алгоритма выработки общего ключа на основе синхронизации обучения искусственных нейронных сетей.

Одним из способов получения двумя абонентами Алисой и Бобом общего ключа является согласованное вычисление ключа в процессе взаимодействия абонентов. Обычно неотъемлемым элементом такого взаимодействия является использование криптографии. Относительно недавно появился новый алгоритм выработки общего ключа, основанный на использовании искусственных нейронных сетей. Искусственным нейронным сетям (ИНС) присущи многие свойства, которые используются в различных прикладных задачах. Например, возможно достижение состояния так называемой синхронизации ИНС, под которой следует понимать равенство значений весовых коэффициентов ИНС. Это явление и может служить основой для протокола выработки общего ключа [1].

Пусть у Алисы и Боба имеется по так называемой древовидной машине чётности (ДМЧ). ДМЧ — это ИНС, которую можно описать следующим образом. В ДМЧ содержится $K$ нейронов скрытого слоя и 1 выходной нейрон. С каждым нейроном скрытого слоя связано $N$ входов, причём с каждым входом связан весовой коэффициент $w[i,j]$ ($i$ – индекс скрытого нейрона, $j$ – индекс входа у скрытого нейрона). Все весовые коэффициенты являются целыми числами, лежащими в диапазоне от $-L$ до $L$. Внутреннее состояние скрытого нейрона определяется как взвешенная сумма его входов, а выход скрытого нейрона — это функция sgn $(x)$ от внутреннего состояния, причём sgn $(x) = -1$ при $x = 0$. Выход выходного нейрона равен произведению выходов нейронов скрытого слоя.

Первоначально значения весов у ДМЧ Алисы и Боба заданы случайно. Значения весовых коэффициентов, внутренние состояния скрытых нейронов и их выходы держатся в секрете в течение всего алгоритма выработки общего ключа. Алгоритм описывается следующим образом.

Алиса и Боб, используя одинаковые входные векторы, вычисляют выходы своих ДМЧ и сообщают их друг другу. При неравных выходных значениях


Омский государственный университет им. Ф.М. Достоевского
E-mail: alizhur@yandex.ru, lavrov@omsu.ru

ДМЧ веса обоих ДМЧ не изменяются, а противном случае применяется некоторое обучающее правило, например обучающее правило Хеббиана:

$$w[i,j] = g(w[i,j] + x[i,j] * out * F(out(i), out) * F(outA, outB)),$$

где $out$ — выход ДМЧ, у которой изменяются веса; $outA$ и $outB$ — это выходы ДМЧ Алисы и Боба соответственно; $x[i,j]$ — вход $j$ скрытого нейрона $i$; $F(x,y) = 1$ при $x = y$, иначе $F(x,y) = 0$;

$$g(x) = \begin{cases} \text{sgn } (x) * L, |x| > L \\ x, |x| \leq L \end{cases}.$$

После некоторого числа итераций обе ДМЧ достигнут состояния синхронизации, причём при проведении итераций и далее весовые коэффициенты, очевидно, будут меняться, но их равенство не нарушится.

При реализации алгоритма возникает вопрос о том, когда следует прекращать его выполнение. Иными словами, требуется найти критерий синхронизации, удовлетворив который, можно будет остановить алгоритм. Возможны следующие подходы:

1. Полный перебор — обеим ДМЧ подаются на входы все возможные входные векторы, а вычисленные выходы ДМЧ сравниваются. При достижении синхронизации при всех входных векторах все соответствующие выходы ДМЧ будут совпадать.

2. Итеративный подход — заключается в эмпирическом оценивании необходимого числа итераций.

3. Использование дайджестов — сравнение Алисой и Бобом дайджестов, вычисленных от набора весов своих ДМЧ. Разумеется, могут быть предложены и другие надёжные криптографические методы для обмена информацией о весах, хотя их применение делает протокол схожим с традиционными аналогами.

В программной реализации у Алисы и Боба использованы по две дополнительные вспомогательные ДМЧ. Так, например, Алиса имеет не только ДМЧ, которая непосредственно участвовала во взаимодействии с Бобом при выработке общего ключа, но и её копия (то есть начальные веса копии и оригинала одинаковы), а также ДМЧ, значения весовых коэффициентов которой инициализированы случайно. Алиса наблюдает за своими вспомогательными ДМЧ и определяет, когда наступит их синхронизация. Далее, при взаимодействии с Бобом Алиса посылает дайджест набора весов Бобу после числа итераций, которое потребовалось для достижения синхронизации вспомогательных ДМЧ Алисы. Аналогично, Боб наблюдает за процессом синхронизации своих ДМЧ и использует полученное при этом число итераций при взаимодействии с Алисой. Участники протокола также могут согласовать перед началом взаимодействия

число итераций, после которых начнётся обмен дайджестами. Описанное применение дополнительных вспомогательных ДМЧ призвано снизить затраты на использование дайджестов.

В качестве источников входных векторов для ДМЧ Алисы и Боба можно использовать одинаковые ГПСП, инициализированные секретным зерном, которое известно лишь Алисе и Бобу. В таком случае может быть исключена передача входных векторов через открытый канал связи, причем возможному злоумышленнику становятся известны только выходы ДМЧ и дайджесты, что увеличивает надежность протокола, так как в описанных атаках на протокол роль Евы заключается в прослушивании взаимодействия Алисы и Боба (предполагается, что Ева не может изменять передаваемую Алисой и Бобом информацию). В ходе такого прослушивания Ева пытается добиться синхронизации одной или нескольких своих ДМЧ с ДМЧ Алисы или Боба, для чего Еве требуется знать и выходы, и входные векторы. В основе стойкости протокола лежит обоснованный факт того, что Алиса и Боб достигают синхронизации быстрее, чем Ева. Кроме того, при использовании ГПСП Алиса и Боб могут аутентифицировать друг друга, так как достижение синхронизации станет признаком знания секретного зерна обеими сторонами.

Достигнув синхронизации ДМЧ, Алиса и Боб могут провести определённые манипуляции над весами и получить из них общий ключ требуемой длины. Например, при $L = 7$ из двух весовых коэффициентов, склеив их, можно легко получить один байт ключа.

Важным моментом является качество получаемых ключей, о котором можно судить, например, по статистическим характеристикам генерируемой последовательности. Проведённые тесты говорят о равномерном распределении ключей.

Надёжность существующих протоколов выработки общего ключа основана, главным образом, на вычислительной сложности задач дискретного логарифмирования и факторизации. Улучшенные версии описанного протокола (не использующие, например, дайджесты весов) могут стать альтернативой применяемым сегодня протоколам.

## Литература

1. Ruttor A. Neural Synchronization and Cryptography. Dissertation zur Erlangung des naturwissenschaftlichen Doktorgrades der Bayerischen Julius-Maximilians-Universit ät Würzburg. Würzburg, 2006. 122 p.

## СВЕДЕНИЯ ОБ АВТОРАХ

**Бесценный Игорь Павлович**, к.ф.-м.н., доцент, кафедра кибернетики, ОмГУ им. Ф.М. Достоевского.
**Бречка Денис Михайлович**, ассистент, кафедра информационной безопасности, ОмГУ им. Ф.М. Достоевского.
**Вишнякова Ольга Анатольевна**, инженер, лаборатория программного обеспечения и компьютерных сетей, ОмГУ им. Ф.М. Достоевского.
**Гусс Святослав Владимирович**, ассистент, кафедра вычислительных систем, ОмГУ им. Ф.М. Достоевского.
**Гуц Александр Константинович**, д.ф.-м.н., профессор, кафедра кибернетики, ОмГУ им. Ф.М. Достоевского.
**Журавлёв Алексей Ильич**, студент, факультет компьютерных наук, ОмГУ им. Ф.М. Достоевского.
**Казанцева Анна Геннадьевна**, студентка, факультет компьютерных наук, ОмГУ им. Ф.М. Достоевского.
**Лавров Дмитрий Николаевич**, к.т.н., доцент, кафедра компьютерных технологий и сетей, ОмГУ им. Ф.М. Достоевского.
**Непомнящих Антон Валерьевич**, аспирант, кафедра компьютерных технологий и сетей, ОмГУ им. Ф.М. Достоевского.
**Непомнящих Егор Валерьевич**, аспирант, кафедра компьютерных технологий и сетей, ОмГУ им. Ф.М. Достоевского.
**Первушин Евгений Александрович**, инженер, кафедра компьютерных технологий и сетей, ОмГУ им. Ф.М. Достоевского.
**Султанкин Евгений Николаевич**, студент, факультет компьютерных наук, ОмГУ им. Ф.М. Достоевского.
**Хлызов Евгений Олегович**, аспирант, кафедра кибернетики, ОмГУ им. Ф.М. Достоевского.

## AUTHORS

**Igor P. Bestsennyi**, Ph.D. (Math.) Associate Professor, Chair of Cybernetics, Omsk State University n.a. F.M. Dostoevskiy.
**Denis M. Brechka**, Post-graduate Student, Chair of Information Security, Omsk State University n.a. F.M. Dostoevskiy.
**Evgeny O. Hlyzov**, Post-graduate Student, Chair of Cybernetics, Omsk State University n.a. F.M. Dostoevskiy.
**Svyatoslav V. Guss**, Post-graduate Student, Chair of Computational System, Omsk State University n.a. F.M. Dostoevskiy.
**Alexander K. Guts**, Doctor of Mathematics, Professor, Chair of Cybernetics, Omsk State University n.a. F.M. Dostoevskiy.
**Anna G. Kazanceva**, Student, Faculty of Computer Sciences, Omsk State University n.a. F.M. Dostoevskiy.
**Dmitry N. Lavrov**, Ph.D.(Eng.) Associate Professor, Chair of Chair of Computer Technology and Networks, Omsk State University n.a. F.M. Dostoevskiy.
**Anton V. Nepomnyaschih**, Post-graduate Student, Chair of Cybernetics, Omsk State University n.a. F.M. Dostoevskiy.
**Egor V. Nepomnyaschih**, Post-graduate Student, Chair of Cybernetics, Omsk State University n.a. F.M. Dostoevskiy.
**Evgeny A. Pervushin**, Engineer, Chair of Computer Technology and Networks, Omsk State University n.a. F.M. Dostoevskiy.
**Evgeny N. Sultankin**, Student, Faculty of Computer Sciences, Omsk State University n.a. F.M. Dostoevskiy.
**Olga A. Vishnyakova**, Engineer of Software and Computers Networks Laboratory, Faculty of Computer Sciences, Omsk State University n.a. F.M. Dostoevskiy.
**Alexei I. Zhuravlev**, Student, Faculty of Computer Sciences, Omsk State University n.a. F.M. Dostoevskiy.

# АННОТАЦИИ

**А.К. Гуц**. Изменения топологии и геометрии пространства, приводящие к образованию кротовой норы.
**Аннотация**. В 3-мерном арифметическом пространстве $\mathbb{R}^3$ задаётся изменяющаяся с течением времени топология, от стандартной евклидовой до некомпактной неодносвязной, описывающей кротовую нору.
**Ключевые слова:** кротовая нора, топология, оценка энергии.

**А.К. Гуц, Е.О. Хлызов**. Модель ярусно-мозаичного леса и моделирование сукцессии.
**Аннотация**. Предлагается математическая модель мозаичного многоярусного леса с целью описания серий сукцессии.
**Ключевые слова:** лесная экосистема, катастрофа «звезда», сукцессия.

**А.Г. Казанцева, Д.Н. Лавров**. Распознавание личности по походке на основе wavelet-параметризации показаний акселерометров.
**Аннотация**. В этой статье предлагается алгоритм распознавания личности по походке на основе wavelet-параметризации показаний акселерометров. Для сбора данных были использованы акселерометры, являющиеся частью устройства SunSPOT. Данные подвергаются предварительной обработке и дальнейшей параметризации на основе вейвлет-декомпозиции. Идентификация походки осуществляется с помощью предварительно обученной искусственной нейронной сети. Компьютерное моделирование даёт точность идентификации около 90%.
**Ключевые слова:** идентификация, походка, акселерометры, SunSPOT, вейвлет, параметризация, распознавание, верификация, биометрия.

**Е.А. Первушин, Д.Н. Лавров**. Использование нормированных кадров сигнала в задаче распознавания по походке.
**Аннотация**. В статье рассматривается задача распознавания человека по показаниям телеметрического датчика во время ходьбы. Описывается метод извлечения признаков, рассматривающий сигнал как периодическую функцию и выделяющий участки, соответствующие периодам. Строится система идентификации на основе метода ближайшего соседа, и приводятся результаты экспериментов.
**Ключевые слова:** распознавание по походке, датчик ускорения, обработка сигналов, определение периода последовательности, метод ближайшего соседа.

**О.А. Вишнякова, Д.Н. Лавров**. Автоматическая сегментация речевого сигнала на базе дискретного вейвлет-преобразования.
**Аннотация**. В данной статье предложен метод сегментации речевого сигнала, основанный на анализе вариации уровня энергии вейвлет-спектра. Расстановка границ происходит на участках быстрого изменения огибающей энергии сигнала сводно по всем уровням детализации.
**Ключевые слова:** сигнал, речь, вейвлет, сегментация, кратномаштабный анализ, фонемы, фрейм, форманта.

**И.П. Бесценный, Е.Н. Султанкин**. Объектно-ориентированный анализ информационной системы для периодических изданий.
**Аннотация**. Обосновывается методология объектно-ориентированного анализа (ООА) и ее применение к информационной системе для периодических изданий.
**Ключевые слова:** объектно-ориентированный анализ, информационные потоки и процессы.

**С.В. Гусс**. Язык детализации каркаса программных компонентов поддержки занятий лингвистической направленности.
**Аннотация**. Для каркаса программных компонентов поддержки занятий лингвистической направленности предлагается предметно-ориентированный язык детализации, позволяющий повысить эффективность разработки и сопровождения конечной программной системы.
**Ключевые слова:** каркас, программные компоненты, модель, разработка, язык детализации.

**А.В. Непомнящих**. Методики приоритизации требований при разработке программного обеспечения.
**Аннотация**. Рассматриваются методики приоритизации требований в проектах по разработке ПО и различные аспекты внедрения данных методик.
**Ключевые слова:** требования, приоритизация, программный проект, технологии разработки.

**Е.В. Непомнящих**. Проектирование пошаговых онлайн-игр.
**Аннотация**. Проводится анализ требований, предъявляемых к современным пошаговым онлайн-играм, и наиболее разумных путей их разработки. Приводится универсальный шаблон базы данных, протокола клиент-серверного взаимодействия и иерархии классов на стороне сервера.
**Ключевые слова:** компьютерные игры, протокол, требования, база данных, шаблоны проектирования, пошаговые стратегии.

**Д.М. Бречка**. Алгоритм поиска мостов типа $\vec{t^*}$ и $\overleftarrow{t^*}$ в графе доступов для дискреционной модели безопасности Take-Grant.
**Аннотация**. В статье приводится алгоритм поиска мостов типа $\vec{t^*}$ и $\overleftarrow{t^*}$ в графе доступов для дискреционной модели безопасности Take-Grant, основанный на классическом алгоритме поиска в ширину.
**Ключевые слова:** безопасность, модель, алгоритм, Take-Grant, мост.

**А.И. Журавлёв, Д.Н. Лавров**. Особенности реализации протокола выработки общего ключа с использованием искусственной нейронной сети.
**Аннотация**. В статье описываются особенности реализации алгоритма выработки общего ключа на основе синхронизации обучения искусственных нейронных сетей.
**Ключевые слова:** общий ключ, криптографический протокол, нейронная сеть.

# ABSTRACTS

**A.K. Guts**. The changes of topology and geometry that create a wormhole.
**Abstract**. In 3-dimensional space the changes of topology and geometry that create a wormhole are considered.
**Keywords:** wormhole, topology, estimate of energy.

**A.K. Guts, E.O. Hlyzov**. Model of a forest ecosystem and succession modelling.
**Abstract**. The mathematical model of a forest ecosystem is offered for the purpose of the description of succession.
**Keywords:** forest ecosystem, star catastrophe, succession.

**A.G. Kazanceva, D.N. Lavrov**. Gait recognition on the basis of wavelet-parametrization of accelerometer indications.
**Abstract**. In this article the algorithm of gait recognition on the basis of wavelet-parametrization of accelerometer indications is offered. For data acquisition the accelerometers which are the part of the SunSPOT device have been used. The data is exposed to preliminary handling and further parametrization, on the basis of wavelet-decomposition. Gait identification is carried out with the help of a pre-trained artificial neural network. Computer simulation produces identification accuracy at about 90 %.
**Keywords:** biometrics, identification, gait, accelerometers, SunSPOT, wavelet, parametrization, recognition, verification.

**E.A. Pervushin, D.N. Lavrov**. Utilization of normalized frames of signal in gait recognition.
**Abstract**. The task of gait recognition from accelerometer data is considered. The article describes feature extraction method that considers the signal as periodical function and extracts frames according to periods. Identification system is built on the basis of the nearest neighbour classifier and experimental results are described.
**Keywords:** gait recognition, accelerometer sensor, signal processing, period detection, nearest neighbour classifier.

**O.A. Vishnyakova, D.N. Lavrov**. Automatic segmentation of a voice signal by the discrete wavelet-transform.
**Abstract**. In the given article the method of segmentation of a voice signal, based on the variation of the level of energy of a wavelet-spectrum is offered. Arrangement of boundaries occurs in sections of fast change of bending energy of a signal on all detail levels.
**Keywords:** signal, speech, wavelet, segmentation, phonemes, frame, formant.

**I.P. Bestsennyi, E.N. Sultankin**. Object-oriented analysis information system for periodicals.
**Abstract**. The methodology of object-oriented analysis is applied to information system for periodicals.
**Keywords:** object-oriented analysis, data flows and processes.

**S.V. Guss**. Detailing language of a frame of program components of support of linguistic-oriented occupations.
**Abstract**. For the frame of program components of support of linguistic-oriented occupations the detailing domain-specific language, allowing to raise the efficiency of development and maintenance for finite program system is offered
**Keywords:** framework, program components, model, development, detailing language.

**A.V. Nepomnyaschih**. Techniques of requirement prioritization at software development.
**Abstract**. Techniques of requirement prioritization in projects on software development and various aspects of the implementation of the given techniques are considered.
**Keywords:** requirement, prioritization, software, techniques, development.

**E.V. Nepomnyaschih**. Designing of step-by-step online games.
**Abstract**. The requirement analysis, shown to the modern step-by-step online games, and the most reasonable ways of their development. The general-purpose template of a database, the protocol Client server interaction and class hierarchy on server side are considered.
**Keywords:** computer games, the protocol, requirements, a database, design pattern, step-by-step strategy.

**D.M. Brechka**. The algorithm of searching for bridges $\vec{t}^*$ and $\overleftarrow{t}^*$ in a protection graph for Take-Grant protection model.
**Abstract**. This article deals with the algorithm of searching for bridges $\vec{t}^*$ and $\overleftarrow{t}^*$ in a protection graph for Take-Grant protection model based on classic algorithm.
**Keywords:** protection, model, algorithm, Take-Grant, bridge.

**A.I. Zhuravlev, D.N. Lavrov**. Features of the protocol implementation of common key generation using an artificial neural network.
**Abstract**. In the article implementation features of the algorithm of common key generation by the synchronization of artificial neural networks are described.
**Keywords:** common key, cryptography protocol, neural network.

# Математические структуры и моделирование

## Выпуск 23

Научный журнал

Главный редактор
**Д.Н. Лавров**

Корректор
**С.Н. Дрозд**

Художественное оформление
**В.В. Коробицын**

---

**Адрес научной редакции**
Россия, 644053, Омск-53, ул. Грозненская, 11
Омский государственный университет
факультет компьютерных наук
E-mail: lavrov@omsu.ru
Электронная версия журнала: http://msm.univer.omsk.su

---

Подписано в печать 28.07.2011. Формат $60 \times 84$ 1/8.
Печ.л. 14,3. Усл. печ. л. 13,3. Уч.-изд.л. 9,4. Тираж 100 экз.

---

Отпечатано на полиграфической базе ООО «Фирма «Банковский сервис»
по заказу ФКН ОмГУ, г. Омск, пр. Маркса 18, корп.1. оф. 304, тел. (3812)510147

ISSN 2222-8772