ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана, ОАО "Концерн "Сириус".

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/pi.html E-mail: prin@novtex.ru

Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

№ 4
April
2014

Published since September 2010

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/pi.html, e-mail:prin@novtex.ru

УДК 004.415

В. И. Шелехов, канд. тех. наук, зав. лаб., Институт систем информатики им. А. П. Ершова, г. Новосибирск,
e-mail: vshel@iis.nsk.su

Язык и технология автоматного программирования

Описана технология автоматного программирования в интеграции с технологиями предикатного и объектно-ориентированного программирования. Автоматная программа реализует конечный автомат в виде гиперграфа управляющих состояний. Технология автоматного программирования иллюстрируется на примерах программы моделирования электронных часов с будильником и протокола передачи данных ATM Adaptation Layer уровня Type 2 AAL.

Ключевые слова: *понимание программ, автоматное программирование, предикатное программирование, уровень адаптации ATM*

V. I. Shelekhov

Automata-Based Software Engineering: the Language and Development Methods

The language and methods of automata-based software engineering integrated with object-oriented and predicate software engineering methods are developed. A program is constructed as a finite state automation presented in the form of control state hypergraph. Automata-based methods are illustrated on the programs of alarm clock modeling and ATM adaptation layer (type 2) protocol.

Keywords: *program comprehension, automata-based programming, predicate programming, ATM Adaptation Layer*

Введение

Автоматная программа определяется в виде конечного автомата и состоит из нескольких сегментов кода. В качестве примера автоматной программы рассмотрим модуль операционной системы (ОС), реализующий сценарий работы с пользователем, показанный на рис. 1.

В управляющем состоянии `login` ОС запрашивает имя пользователя. Если полученное от пользователя имя существует в системе, она переходит в управляющее состояние `passw`, иначе возвращается в состояние `login`. В состоянии `passw` система запрашивает пароль. Если поданная пользователем строка соответствует правильному паролю, то пользователь допускается к работе и система переходит в состояние `session`. При завершении работы пользователя система переходит в состояние `login`.

Всякая вершина автомата соответствует некоторому *управляющему состоянию*. Ориентированная гипер-

```
login:   if (exists(get_string()))
             goto passw
         else  goto login
passw:   if (valid(get_string()))
             goto session
         else  goto passw
session: <сессия пользователя>;
         goto login
```

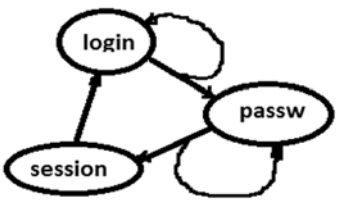

Рис. 1. Схема работы ОС с пользователем

дуга автомата соответствует некоторому *сегменту кода* и связывает одну вершину с одной или несколькими другими вершинами. Исполнение сегмента завершается оператором перехода на начало другого сегмента.

Понятие автоматной программы концептуально не отличается от введенного Анатолием Шалыто [1], однако различия в языке и технологии существенны.

Определим следующие классы программ.

Класс *невзаимодействующих программ* или класс *программ-функций*. Программа принадлежит этому классу, если она не взаимодействует с внешним окружением. Точнее, если возможно перестроить программу таким образом, чтобы все операторы ввода данных находились в начале программы, а весь вывод собран в конце программы, то такая программа относится к классу невзаимодействующих программ. Программа обязана всегда завершаться, поскольку бесконечно работающая и невзаимодействующая программа бесполезна. Следовательно, программа определяет функцию, вычисляющую по набору входных данных (аргументов) некоторый набор результатов. Класс программ-функций по меньшей мере содержит программы для задач дискретной и вычислительной математики.

Программа-функция реализует решение (алгоритм) некоторой математической *задачи*. Решение задачи строится на базе *логики решения* — набора математических свойств (теорем). Программирование — это реализация логики решения в конструкциях языка программирования. Понимание программы реализуется в процессе сопоставления кода программы с логикой решения. В качестве иллюстрации рассмотрим логику решения для простейшей программы умножения натуральных чисел a и b через операцию сложения:

```
a * b = b + (a − 1) * b, 0 * b = 0.        (1)
```

Ниже приведены логическая, функциональная и императивная программы, являющиеся реализациями логики решения (1).

```
0 * b → 0; a * b → b + (a − 1) * b          (2)
```

```
nat mult(nat a, b)                          (3)
   { if (a = 0) 0 else b + mult(a − 1, b) }
```

```
nat c = 0;                                  (4)
while (a ! = 0) { c = c + b; a = a − 1 }
```

Различия логической программы (2) и логики (1) чисто синтаксические. Фактически они тождественны. Функциональная программа (3) и логика (1) отличаются. Однако они также тождественны — программа (3) легко транслируется в логику (1). Императивная программа (4) также построена из логики (1), однако понять ее и установить тождественность с логикой (1) — нетривиальная задача в силу трудности понимания циклов типа **while**[1].

[1] Косвенным подтверждением является чрезвычайная трудность автоматического построения инвариантов циклов.

Класс *программ-процессов* (*автоматных программ*). Программа данного класса является автоматной программой, состоящей из набора сегментов кода. Всякий сегмент есть либо программа-функция, либо программа-процесс, декомпозиция которой представлена другим автоматом. Исполнение автоматной программы может быть бесконечным процессом. Взаимодействие с внешним окружением автоматной программы реализуется через прием и посылку *сообщений*, а также через *разделяемые переменные*, доступные в данной программе и в других программах из окружения программы. Операторы *ввода* и *вывода* рассматриваются как упрощенная форма операторов посылки и приема сообщений. *Состояние* автоматной программы определяется значениями набора переменных, модифицируемых в программе, за исключением локальных переменных. Управляющее состояние программы идентифицирует текущий исполняемый сегмент.

Важнейшим подклассом автоматных программ являются контроллеры систем управления в аэрокосмической отрасли, в энергетике, в медицине, в массовом транспорте и в других отраслях. На каждом шаге вычислительного цикла контроллер получает входную информацию из окружения и обрабатывает ее. Результаты вычисления используются для передачи управляющего сигнала в окружение контроллера. Значительная часть *программируемых логических контроллеров* PLCs (*programmable logic controllers*) разрабатывается в соответствии со стандартом IEC 61131-3.

Отметим, что классы программ-функций и программ-процессов не охватывают всего множества программ. Например, компиляторы и ОС относятся к более сложным классам программ.

Автоматные программы по своей структуре существенно сложнее программ-функций. Автоматное программирование разработано для программ-процессов, и его применение не оправдано для программ-функций. Технология автоматного программирования должна быть интегрирована с технологиями для класса программ-функций, поскольку автоматная программа строится из программ-функций.

В целях улучшения понимания автоматных программ одной из задач технологии автоматного программирования является применение методов, позволяющих упростить сегменты кода автоматной программы. В частности, здесь применимы методы объектно-ориентированного и предикатного программирования: введение объектов вместо наборов переменных позволяет разгрузить автоматную программу, локализуя часть связей внутри классов, а использование функциональных (предикатных) программ вместо аналогичных императивных программ позволяет существенно упростить программу.

Общее описание парадигмы предикатного программирования дано в разд. 1 настоящей статьи. Базисные конструкции языка автоматного программирования определены в разд. 2. На примере программы моделирования электронных часов с будильником в

разд. 3 определен базис технологии автоматного программирования. В разд. 4 технология автоматного программирования проиллюстрирована на примере сложного протокола передачи данных ATM Adaptation Layer уровня Type 2 AAL [2]. Обзор смежных работ представлен в разд. 5. В заключении определены задачи развития предложенного подхода для различных подклассов программ-процессов.

1. Предикатное программирование

Предикатная программа относится к классу программ-функций и является предикатом в форме вычислимого оператора, адекватно представляющим логику решения задачи, обеспечивая хорошее понимание программы. Язык предикатного программирования P [3] обладает большей выразительностью в сравнении с языками функционального программирования и по стилю ближе к императивному программированию.

Предикатная программа состоит из набора рекурсивных программ (определений предикатов) на языке P следующего вида:

```
<имя программы>(<описания аргументов> :
                <описания результатов>)
  pre <предусловие>
  { <оператор> }
  post <постусловие>
```

Необязательные конструкции предусловие и постусловие являются формулами на языке исчисления предикатов; их используют для улучшения понимания программ и для дедуктивной верификации [4—7]. Ниже представлены основные конструкции языка P: оператор присваивания, блок (оператор суперпозиции), параллельный оператор, условный оператор, вызов программы и описание переменных, используемое для аргументов, результатов и локальных переменных.

```
<переменная> = <выражение>
{<оператор1>; <оператор2>}
<оператор1> || <оператор2>
if (<логическое выражение>) <оператор1>
else <оператор2>
<имя программы>(<список аргументов>:
                <список результатов>)
<тип> <пробел> <список имен переменных>
```

В предикатном программировании запрещены такие языковые конструкции, как циклы и указатели, серьезно усложняющие программу. Известно, что функциональная программа в несколько раз проще в сравнении с императивной программой, реализующей тот же алгоритм, потому что вместо циклов используют рекурсивные функции, а вместо массивов и указателей — списки.

Эффективность предикатных программ достигается применением следующих оптимизирующих трансформаций, переводящих программу на императивное расширение языка P:

- замена хвостовой рекурсии циклом;
- подстановка тела программы на место ее вызова;
- склеивание переменных: замена всех вхождений одной переменной на другую переменную;
- кодирование алгебраических типов (списков и деревьев) с помощью массивов и указателей.

Эффективность программы после применения трансформаций обеспечивается оптимизацией, реализуемой программистом, на уровне предикатной программы. Для приведения рекурсии к хвостовому виду применяется метод обобщения исходной задачи. Далее обычно открывается возможность проведения серии последующих улучшений алгоритма. Итоговая программа по эффективности не уступает написанной вручную и, как правило, короче [4—8]. Отметим, что в функциональном программировании (при общеизвестной ориентации на предельную компактность и декларативность [9]) оптимизация программы полностью возлагается на транслятор, в частности, обеспечивается автоматическое приведение рекурсии к хвостовому виду. Разумеется, функциональное программирование существенно уступает в эффективности, поскольку даже применением изощренных методов оптимизации невозможно автоматически воспроизвести серию оптимизаций, совершаемых программистом вручную.

Гиперфункции. Рассмотрим типичное решение задачи: взять второй элемент списка s и обработать его программой B; если второго элемента нет, запустить программу D:

```
Elem2(s: e, exists);                                (5)
if (exists) B(e...) else D(...);
```

Программа Elem2 на языке P:

```
pred Elem2(list(int) s: int e, bool exists)
{if (s = nil ∨ s.cdr = nil) exists = false
 else { e = s.cdr.car || exists = true }
} post exists = (s ≠ nil & s.cdr ≠ nil) &
                (exists ⇒ e = s.cdr.car);
```

Использование логической переменной exists для реализации ветвления в (5) — плохое решение, загромождающее и усложняющее программу. Золотое правило программирования[2] состоит в том, чтобы не использовать логических переменных для реализации

[2]Геннадий Кожухин, разработчик транслятора АЛЬФА, собирал золотые правила программирования в 1960-х гг.

ветвления в программе[3]. Правильным решением является следующая *гиперфункция* для Elem2:

```
hyper Elem2(list (int) s : int e #1: #2)
pre 1: s ≠ nil & s.cdr ≠ nil
{if (s = nil ∨ s.cdr = nil) #2
 else { e = s.cdr.car; #1 }
} post 1: e = s.cdr.car;
```

Гиперфункция Elem2 имеет две *ветви* результатов: первая ветвь включает переменную e, а вторая ветвь не имеет результатов — она пуста. *Метки* 1 и 2 — дополнительные параметры, обозначающие два различных *выхода* гиперфункции. *Спецификация гиперфункции* состоит из двух частей. Утверждение после "**pre** 1:" есть предусловие первой ветви; предусловие второй ветви — просто отрицание предусловия первой ветви. Утверждение после "**post** 1": есть постусловие для первой ветви. Фрагмент (5) заменяется следующим:

```
Elem2(s: e #L1: #L2)
            case L1: B(e...)
            case L2: D(...);
```

Исполнение вызова гиперфункции завершается переходом либо по метке L1, либо по метке L2. Операторы перехода #L1 и #L2 внутри вызова гиперфункции явным образом фиксируют точки передачи управления в отличие от вызова вида Elem2(s, e, L1, L2) на языке типа Алгол-60, где переходы по меткам L1 и L2 синтаксически не выделены и их можно не заметить. Эта одна из причин, почему оператор перехода **goto** L1 записывается в виде #L1.

Чтобы дать общее определение гиперфункции, рассмотрим следующее определение предиката:

```
pred A(x: y, z, c)                                   (6)
pre P(x)
{ ... }
post c = C(x) & (C(x) ⇒ S(x, y)) &
                (¬C(x) ⇒ R(x, z));
```

Здесь x, y и z — возможно пустые наборы переменных; P(x), C(x), S(x, y) и R(x, z) — логические утверждения. Предположим, что все присваивания вида c = **true** и c = **false** — последние исполняемые операторы в теле предиката. Определение предиката (6) может быть заменено следующим определением гиперфункции:

```
hyp A(x: y #1: z #2)                                 (7)
pre P(x) pre 1: C(x)
{ ... }
post 1: S(x, y) post 2: R(x, z);
```

В теле гиперфункции каждое присваивание c = **true** заменено на #1, а c = **false** — на #2.

Допустим, имеется условный оператор вида

```
if (C) {A; B} else {E; D},                           (8)
```

где A, B, E и D — некоторые операторы. Определим гиперфункцию:

```
hyper H(...: ...#1: ...#2) pre 1:C
{ if (C) {A; #1} else {E; #2} }.
```

Тогда условный оператор (8) эквивалентен следующей композиции:

```
H(...: ...#1: ...#2) case 1: B case 2: D.
```

Данное свойство демонстрирует, что аппарат гиперфункций позволяет проводить гибкую декомпозицию программы, недостижимую другими средствами [4, 7, 10]. Аппарат гиперфункций является более общим в сравнении с обработкой исключений в языках Java и C#. Гиперфункции не представимы монадами[4] функциональных языков. Использование гиперфункций делает программу короче, быстрее и проще для понимания [4, 7, 10]. Отметим, что модель программ-процессов в форме гиперграфа является естественным продолжением аппарата гиперфункций.

2. Язык автоматного программирования

Ограничимся рассмотрением простых процессов без параллелизма и недетерминизма. Язык автоматного программирования строится расширением языка для класса программ-функций. Программа-процесс определяется следующей конструкцией:

```
process <имя программы>
            (<описания аргументов и
             результатов>)
pre <предусловие>
  { <описания переменных состояния процесса>
    <сегменты кода>
  }
```

Произвольный <сегмент кода> представляется следующим образом:

```
<имя управляющего состояния>:
        inv <инвариант сегмента>;
        <оператор>;
```

<Инвариант сегмента> должен быть истинным перед выполнением <оператора>. Инвариант не является обязательным, он используется для улучшения понимания автоматной программы и для верификации. Исполнение <оператора> завершается исполнением оператора перехода на начало другого сегмента автоматной программы. При нормальном заверше-

[3] Это противоречит принципу структурного программирования, поскольку структурирование произвольной программы в соответствии с теоремой Боэма и Джакопини реализуется с использований дополнительных логических переменных.

[4] Возможности, определяемые монадами, явным образом предоставляются в языке P.

нии <оператора> исполнение продолжится с начала следующего сегмента.

Для взаимодействия с внешним окружением используют средства ввода и вывода данных, посылки и приема сообщений с возможными параметрами. Оператор **send** m(e) посылает сообщение m с параметрами, значения которых определяются набором выражений e. Оператор приема сообщений определяется следующим правилом:

```
<оператор приема сообщений>:: =
   receive <имя сообщения>(<параметры>)
                              <оператор> |
   receive <имя сообщения> (<параметры>)
                              { <оператор> }
   else <оператор>
```

Для второго варианта правила при отсутствии сообщения в канале исполняется **else**-часть оператора, возможно содержащая прием других сообщений. Если все альтернативы определяют прием сообщений и все эти сообщения отсутствуют, <оператор приема сообщений> исполняется многократно до тех пор, пока в канале не появится одно из сообщений, принимаемых оператором. После получения сообщения переменные, указанные <параметрами>, получают значения параметров сообщения), и выполняется соответствующий <оператор>.

3. Электронные часы с будильником

На примере автоматной программы "Электронные часы с будильником" [1] дадим иллюстрацию технологии автоматного программирования, интегрированной здесь с объектно-ориентированной технологией. Язык предикатного программирования P расширяется описаниями классов и конструкцией <объект>.<имя элемента класса> для доступа к элементу некоторого объекта. Описание класса определяет переменные, константы и методы как элементы класса. Описание метода представляется в виде определения предиката.

Рассмотрим устройство электронных часов с будильником [1]. На корпусе часов три кнопки:

- H (*Hours*) — увеличивает на единицу число часов;
- M (*Minutes*) — увеличивает на единицу число минут;
- A (*Alarm*) — включает и выключает будильник.

Увеличение часов и минут происходит по модулю 24 и 60 соответственно. Если будильник выключен, то кнопка A включает его и переводит часы в режим, в котором кнопки H и M устанавливают не текущее время, а время срабатывания будильника. Повторное нажатие кнопки A переводит часы в режим с включенным будильником, в котором кнопки H и M будут менять время на часах. В режиме с включенным будильником, если текущее время совпадает со временем будильника, включается звонок, который отключается либо нажатием кнопки A, либо самопроизвольно через минуту. Нажатие кнопки A в режиме с включенным будильником переводит часы в нормальный режим без будильника.

Набор различных действий с часами, встречающихся в автоматной программе моделирования функционирования часов, представим в виде класса Часы.

```
class Часы {
  nat hours, minutes;
        // текущее время (часы, минуты)
  nat alarm_hours, alarm_minutes;
        // время срабатывания будильника
  inc_h(); {...}
        // увеличить время на один час
  inc_m(); {...}
        // увеличить время на одну минуту
  inc_alarm_h(); {...}
        // увеличить время  будильника на час
  inc_alarm_m(); {...}
        // увеличить время будильника на минуту
  tick() {...} // приращение времени на тик –
               // минимальный интервал
  bell_on() {...} // включить звонок
  bell_off() {...} // выключить звонок
  bool bell_limit() {...};
        // звонок звонит уже минуту
}
```

Использование класса Часы позволяет существенно разгрузить и тем самым упростить автоматную программу. В качестве состояния автоматной программы используется одна переменная t (объект класса Часы), вместо четырех переменных, которые были бы использованы в версии автоматной программы без применения объектно-ориентированной технологии. Вместе с четырьмя переменными класс Часы также локализует внутри себя набор условий, формулируемых обычно в виде инвариантов, а также предусловий и постусловий методов.

Имеются три управляющих состояния автоматной программы:

- off — режим работы часов без будильника;
- set — режим установки будильника;
- on — режим работы часов с включенным будильником.

Представленная ниже автоматная программа является непосредственной формализацией данного выше содержательного описания функционирования часов. Разумеется, было бы правильным реализовать естественный ход часов независимым параллельным процессом. Данный алгоритм выбран, потому что он ближе к алгоритму, описанному в работе [1].

```
process Работа_часов_с_будильником {                                          (9)
  Часы t = Часы();
  off: receive H { t.inc_h() #off }                                           (9.1)
       else receive M { t.inc_m() #off }
       else receive A { #set }                                                (9.2)
       else { tick() #off }
  set: receive H { t.inc_alarm_h() #set }
       else receive M { t.inc_alarm_m() #set }
       else receive A { t.bell_on() #on }
       else { tick() #set }
  on:  receive H { t.inc_h() #on }
       else receive M { t.inc_m() #on }
       else receive A { t.bell_off() #off }
       else { tick(); if (t.bell_limit()) {t.bell_off() #off } else #on }     (9.3)
}
```

Сравним данную программу с аналогичной, описанной в работе [1]. С этой целью трансформируем ее так, чтобы получить программу в соответствии со switch-технологией А.А. Шалыто [11]. Для множества управляющих состояний введем описание типа:

```
type State = enum(off, set, on).
```

Введем переменную `state`, определяющую значение текущего управляющего состояния. Заменим операторы перехода на соответствующие присваивания переменной `state`, например, переход `#off` заменим на `state = off`; далее часть таких присваиваний можно опустить в случаях, когда управляющее состояние не меняется. Для реализации переходов на нужные управляющие состояния используем оператор **`switch`** по значению переменной `state`. Наконец, представим итоговую программу в виде бесконечного цикла.

```
process Работа_часов_с_будильником {                                          (10)
  Часы t = Часы();
  State state = off;
  while (true)
  switch (state) {
     case off: receive H      { t.inc_h() }                                   (10.1)
               else receive M { t.inc_m() }
               else receive A { state = set }                                 (10.2)
               else { tick() }
     case set: receive H { t.inc_alarm_h() }                                 (10.3)
               else receive M  { t.inc_alarm_m() }
               else receive A  { t.bell_on(); state = on }
               else { tick() }
     case on:  receive H { t.inc_h() }
               else receive M  { t.inc_m() }
               else receive A  { t.bell_off(); state = off }
               else { tick(); if (t.bell_limit()) { t.bell_off(); state = off } }   (10.4)
   }}
```

Программа (10) похожа на одну из версий этой программы, описанную в работе [1, п. 2.3.2, с. 92]. На рис. 2 приведено сравнение программ (9) и (10). В соответствии с присваиванием `state = set` в строке (10.2) дальнейшее управление через оператор **`switch`** будет передано на строку (10.3). Аналогичный переход в программе (9) реализуется явным образом оператором `#set` в строке (9.2). Поскольку в строке (10.1) нет присваиваний переменной `state`, она сохраняет прежнее значение `off`; поэтому в соответствии с оператором **`switch`** управление будет передано снова на строку (10.1). Аналогичный переход в программе (9) реализуется явно оператором `#off` в строке (9.1). Таким образом, программа (9) проще, поскольку все переходы в ней реализуются явно операторами перехода, тогда как аналогичные переходы в программе (10) опосредованы через присваивания переменной `state` и механизм действия оператора **`switch`**.

Последний сегмент программы (9) имеет структуру гипердуги. Чтобы привести программу (9) в соответствие с классическим конечным автоматом, необходимо добавить четвертое управляющее состояние после вызова `tick()` на строке (9.3). В программе, представленной листингом 2.8 [1, п. 2.3.2, с. 92], удается избежать добавления управляющего состояния, однако сложным и неадекватным образом, подобно втягиванию вызова `tick()` внутрь условного оператора на строке (10.4) с добавлением в условном операторе

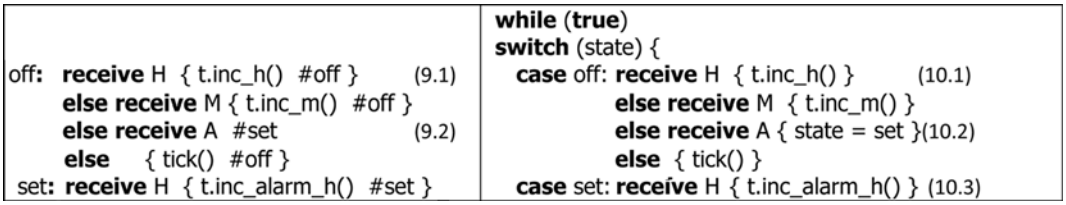

Рис. 2. Сравнение программ (9) и (10) на начальном сегменте кода

еще двух ветвей. Это лишь один показательный пример того, что используемая в рассматриваемом подходе гиперграфовая структура обладает лучшими выразительными возможностями в сравнении со структурой классического конечного автомата.

Сформулируем требования, составляющие дисциплину автоматного программирования. Во-первых, в автоматной программе переходы разрешены лишь на начало одного из сегментов, либо на метку ветви завершения процесса, указанную в <описании аргументов и результатов>. Во-вторых, все сегменты автоматной программы (а значит и все переходы автоматной программы) должны быть одновременно визуально доступны. Это важное эргономическое требование, означающее, что автоматная программа должна полностью помещаться на экране дисплея. Требуемое сокращение размера автоматной программы, в частности, достигается переносом ее частей в методы или подпрограммы. В предельном случае сегмент кода может быть заменен соответствующим вызовом гиперфункции. Например, первый сегмент программы (9) может быть замене на

```
off: Шаг_часов_без_будильника (t: #of: #set).
```

Более радикальной является декомпозиция автоматной программы, при которой ее части определяются независимыми процессами. Например, возможна следующая декомпозиция программы (9).

```
process Часы_без_будильника (Часы t: # set) {
   1:   receive H { t.inc_h() #1 }
        else receive M { t.inc_m() #1 }
        else receive A #set
        else { tick() #1 }
}
process Установка_будильника (Часы t: # on) {
   set: receive H { t.inc_alarm_h() #set }
        else receive M { t.inc_alarm_m() #set }
        else receive A { t.bell_on() #on }
        else { tick() #set }
}
process Часы_с_установленным_будильником (Часы t: # off) {
   3:   receive H { t.inc_h() #3 }
        else receive M { t.inc_m() #3 }
        else receive A { t.bell_off() #off }
        else { tick(); if (t.bell_limit()) { t.bell_off() #off } else #3 }
}
process Работа_часов_с_будильником {
   Часы t = Часы();
   off: Часы_без_будильника (t: # set);
   set: Установка_будильника (t: #on);
   on: Часы_с_установленным_будильником (t: #off)
}
```

Возможен другой способ декомпозиции: любая пара сегментов программы (9) может быть объединена в единый независимый процесс. Пример:

```
process Работа_часов_с_будильником {
   Часы t = Часы();
   off: Часы_без_будильника_Установка_будильника (t: #on);
   on: Часы_с_установленным_будильником (t: # off)
}
```

Процесс из одного сегмента кода можно было бы представить оператором цикла, например, следующим образом:

```
process Часы_без_будильника (Часы t: # set) {
   loop { receive H t.inc_h()
          else receive M t.inc_m()
          else receive A #set
          else tick()
}}
```

Тем не менее запрещено использование операторов цикла для описания процессов в автоматной программе. Они, безусловно, сокращают размер кода, однако не улучшают понимания программы. Использование операторов цикла неприемлемо для декомпозиции процессов в автоматных программах и в отдельных случаях увеличивает число управляющих состояний.

4. Передатчик в протоколе AAL-2

Технология автоматного программирования иллюстрируется на примере программы `Передатчик` в протоколе AAL-2, описанной на языке спецификаций SDL [12] в подразделе 10.1 стандарта [2]. В представленной далее версии программы опущена детальная информация по протоколу напрямую не связанная с алгоритмом передачи данных, при этом большая часть обозначений сохранена. Использование аппарата гиперфункций, а также списков вместо массивов позволяет существенно упростить программу `Передатчик`. Однако она остается достаточно сложной. Поэтому применяется двухуровневый подход к ее построению на базе скелета, получаемого из предикатной программы передачи пакетов данных с плотной упаковкой в формате блоков постоянной длины.

Уровень адаптации 2 (AAL type 2) в асинхронном способе передачи данных ATM (*Asynchronous Transfer Mode*) применяется для эффективной передачи низкоскоростных, коротких пакетов переменной длины в приложениях, чувствительных к задержкам. Используют три уровня передачи данных: подуровень конвергенции SSCS (*Service Specific Convergence Sublayer*), общий подуровень CPS (*Common Part Sublayer*) и уровень ATM. `Передатчик` получает пакеты с уровня SSCS, плотно пакует их в один или несколько блоков данных и посылает на уровень ATM.

`Передатчик` получает очередной пакет через сообщение `CPSpacket(pd, atrs)`, где `pd` — данные пакета переменной длины, `atrs` — атрибуты пакета. `Передатчик` строит *заголовок пакета* `ph` (*packet header*) длиной три октета[5]. Пакет, состоящий из `ph` и `pd`, пакуется в один или несколько блоков данных PDUs (*protocol data units*). Блок данных PDU состоит из *начального поля* `STF` (*start field*) длиной в один октет и *основной части* (*payload*) длиной 47 октетов. Пакет, не вместившийся в очередном блоке PDU, продолжается в следующем PDU. Начальное поле `STF` блока PDU содержит длину в октетах перенесенной части пакета из предыдущего блока PDU. Очередной построенный блок PDU передается на уровень ATM посылкой сообщения `ATM_data(PDU)`.

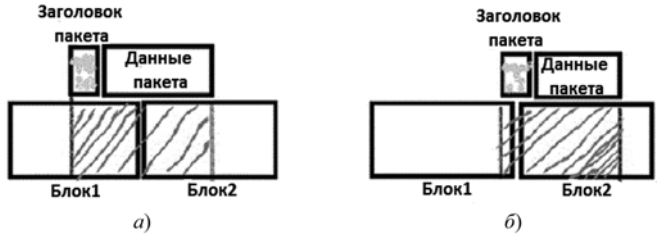

Рис. 3. Схема переноса пакета:
а — перенос данных; *б* — перенос заголовка

Упаковка пакетов в блоке PDU плотная. Следующий пакет размещается в блоке PDU непосредственно после предыдущего. Блок PDU может содержать несколько пакетов. Заголовок `ph` также может быть перенесен из одного блока в другой (рис. 3).

Остальную информацию по программе `Передатчик` представим ниже.

Построим программу `transfer`, являющуюся скелетом программы `Передатчик` и реализующую передачу пакетов с плотной упаковкой в последовательности блоков PDU. Очередной блок PDU формируется в буфере `buf` длины 48 октетов. Для переписи на буфер `buf` произвольного пакета `x` произвольной длины применяется гиперфункция `move`. Предварительно дадим необходимые описания программы:

```
type OCTET = byte;
type DATA = list(OCTET);
type ATRS; // атрибуты пакета
```

Тип `DATA` определяет списки, элементами которого являются октеты.

```
pred split(DATA x, nat r: DATA y, x')
     pre len(x) ⩾ r
     post x = y + x' & len(y) = r;
```

Здесь обозначение `x'` используется для итогового значения `x`, модифицированного программой. Операция + означает конкатенацию двух списков. Программа `split` извлекает список `y` длины `r` из начала списка `x`; при этом `x'` — оставшаяся часть списка.

Гиперфункция `move` добавляет список `x` к списку `buf`. Первая ветвь гиперфункции соответствует случаю, когда `x` по длине помещается в оставшуюся часть буфера `buf`.

[5] Октет состоит из 8 битов; термин "байт" не используется, так как имеет иной смысл.

```
hyper move(DATA x, buf : DATA x′, buf′ #1 : DATA x′, buf′ #2)
  pre len(buf) ⩽ 48 pre 1: len(buf) + len(x) ⩽ 48
{ if (len(buf) + len(x) > 48) { split(x, 48 − len(buf): DATA y, x′); buf′ = buf + y #2 }
  else { buf′ = buf + x; x′ = nil #1 }
} post 1: len(buf′) ⩽ 48 & buf′ = buf + x & x′ = nil
  post 2: len(buf′) = 48 & buf + x = buf′ + x′;
```

Если длина `buf + x` больше `48` (ветвь 2), список `x` расщепляется на `y` и `x′`, причем `y` дописывается в список `buf`, а `x′` — оставшаяся часть от списка `x`, т. е. `len(buf + y) = 48` и `x = y + x′`. Заметим, что присваивание `x′ = nil` на первой ветви необходимо, потому что переменная `x′` включена в состояние процесса программы `Передатчик`; в противном случае `x′` можно было бы изъять из результатов первой ветви, что упростило бы алгоритм.

Чтобы обеспечить эффективность программы в трансляторе с языка `P` реализуется трансформация замены списков массивами. Список `x` кодируется внутри массива `X` в виде вырезки массива `X[jx: px-1]`; `buf` кодируется вырезкой `BUF[0: pBuf-1]`. Проводятся следующие замены: `len(x)` → `px − jx`, `len(buf)` → `pBuf`. Ниже представлен код гиперфункции после трансформации.

```
hyper move(X, jx, px, BUF, pBuf : jx′, px′, pBuf′ #1 : jx′, px′, pBuf′ #2)
{ if (pBuf + px − jx > 48) { nat py = 48 + jx − pBuf; BUF[pBuf: 48] = X[jx: py-1];
                             pBuf = pBuf + py − jx; jx′ = py #2 }
  else { BUF[pBuf: ] = X[jx: px-1]; pBuf = pBuf + px − jx; jx′ = px #1 }
};
```

Определим типы данных программы `transfer` и предописания функций для создания заголовков `STF` и `ph`:

```
type BLOCK = DATA; // блок длиной 48 октетов
type INPUT = list(DATA); // тип последовательности пакетов
type OUT = list(BLOCK); // тип последовательности блоков
OCTET ConstructSTF(nat len_of_packet_continuation);
DATA ConstructCPSPacketHeader(DATA pd, ATRS atrs);
```

Предикатная программа `transfer` преобразует потенциально бесконечную последовательность `in` пакетов типа `INPUT` в последовательность `out` блоков типа `OUT` с плотной упаковкой. Поскольку список `in` никогда не завершается, можно упростить программу удалением ее ветвей для случая `in = nil`:

```
pred transfer(INPUT in: OUT out)
{ transfer1(in, ConstructSTF(0), nil: out) }
```

Программа `transfer` сводится к более общей программе `transfer1`, у которой второй параметр `buf` — текущий буфер, а третий параметр `out0` — ранее сформированная последовательность блоков. В вызове `transfer1` в качестве начального состояния буфера `buf` определяется октет `STF` с нулевой длиной перенесенной части, поскольку предыдущий пакет отсутствует.

```
pred transfer1(INPUT in, BLOCK buf, OUT out0: OUT out)
{ DATA pd = in.car, ph = ConstructCPS_PacketHeader(pd);
  transfer2(in, ph, pd, buf, out0: out)
}
```

В программе `transfer1` из входного потока `in` извлекается очередной пакет `pd`, и конструируется заголовок пакета `ph`. Пакет `pd` и заголовок `ph` — дополнительные параметры более общей программы `transfer2`. Параметры `pd` и `ph` программы `transfer2` определяют те части пакета и заголовка, которые еще не переписаны на буфер `buf`. В частности, если заголовок уже переписан, то `ph = nil`.

```
pred transfer2(INPUT in, DATA ph, pd, BLOCK buf, OUT out0: OUT out)
{ { move(ph, buf: ph′, buf′ #G: ph′, buf′ #B);
   G: move(pd, buf: pd′, buf′ #Z: pd′, buf′ #B);
   Z: transfer1(in.cdr, buf′, out0: out)
   } case B: { BLOCK buf1 =
                 ConstructSTF((len(ph) + len(pd) > 47)? 47 : len(ph) + len(pd));
               transfer2(in, ph, pd, buf1, out0 + buf: out)
             }
}}
```

Используя вызовы гиперфункции move, программа transfer2 переписывает на буфер buf сначала заголовок ph, а затем пакет pd. Если оба вызова завершились по первой ветви, то передача текущего пакета завершена. Передача следующих пакетов реализуется вызовом программы transfer1. Если один из вызовов гиперфункции move завершился по второй ветви, срабатывает обработчик с меткой B. Конструируется новый буфер buf1 с начальным октетом STF, в котором фиксируется длина переносимой части текущего пакета, точнее оставшиеся (еще не переписанные) части заголовка ph и пакета pd. Оставшиеся части ph и pd переписываются в новый буфер buf1 вызовом программы transfer2, а заполненный буфер buf поступает в выходную последовательность out.

Проведем следующие трансформации предикатной программы transfer. Реализуем склеивание out0 → out в программе transfer1 и набор склеиваний в программе transfer2: out0 → out; ph′ → ph; buf1, buf′ → buf. Далее подставим тело программы transfer1 на место вызова в программе transfer2. Хвостовая рекурсия в transfer2 позволяет заменить рекурсию циклом. Тело программы transfer2 подставляется на место вызова в transfer1. Наконец, transfer1 подставляется в transfer. Операции с входными и выходными потоками in и out заменяют соответствующими операторами посылки и приема сообщений:

```
in = in.cdr; pd = in.car →
                    receive CPSpacket(pd)

out = out + buf → send ATM_data(buf)
```

Замена списков ph, pd и buf массивами пока не проводится. Программа transfer после трансформаций принимает вид следующей автоматной программы:

```
process transfer()                                                                    (11)
{   BLOCK buf = ConstructSTF(0);
    receive CPSpacket(pd);                                                            (11.1)
    ph = ConstructCPS_PacketHeader(pd);                                               (11.2)
H:  move(ph, buf: ph, buf: ph, buf #B);
    move(pd, buf: pd, buf: pd, buf #B);
    receive CPSpacket(pd);                                                            (11.3)
    ph = ConstructCPS_PacketHeader(pd);                                               (11.4)
    #H                                                                                (11.5)
B:  send ATM_data(buf);
    buf1 = ConstructSTF((len(ph) + len(pd) > 47)? 47 : len(ph) + len(pd));
    #H                                                                                (11.6)
}
```

В первой ветви вызовов гиперфункции move опущены операторы перехода, поскольку переход реализуется на следующий оператор. Чтобы избежать совпадения операторов (11.1) и (11.2) с (11.3) и (11.4) в программе (12) для оператора (11.1) заведена метка IDLE. Вставка оператора (12.2) и замена (11.6) на (12.3)—(12.5) определяют оптимизации, присутствующие в исходной программе Передатчик [2].

```
process transfer()                                                                    (12)
{       BLOCK buf = ConstructSTF(0);
IDLE:   receive CPSpacket(pd);                                                        (12.1)
        ph = ConstructCPS_PacketHeader(pd);
H:      move(ph, buf: ph, buf: ph, buf #B);
G:      move(pd, buf: pd, buf: pd, buf #B);
        if (len(buf) = 48) #B;                                                        (12.2)
        #IDLE
B:      send ATM_data(buf);
        buf = ConstructSTF((len(ph) + len(pd) > 47)? 47 : len(ph) + len(pd));
        if (ph ≠ nil) #H                                                              (12.3)
        else if (pd = nil) #IDLE                                                      (12.4)
        else #G                                                                       (12.5)
}
```

Программа Передатчик, представленная на рис. 4, построена модификацией программы (12) с учетом следующих дополнительных требований. Очередной заполненный блок PDU может быть отправлен на уровень ATM только после получения запроса на посылку — сообщения SEND_request(). Построение очередного блока PDU управляется таймером. При истечении установленного времени построение очередного блока PDU завершается заполнением нулями оставшейся части блока PDU вызовом программы fill со следующей спецификацией:

```
START: permit := false;
       buf = ConstructSTF(0);
IDLE: inv pd = nil & len(buf) = 1;
       receive CPSpacket(pd, ATRS atrs) {
         set Timer_CU;
C:       inv len(buf) < 48 & pd≠nil & ph=nil;
         ph=ConstructCPS_PacketHeader
                                  (pd, atrs);
H:       inv len(buf) < 48 & pd≠nil & ph≠nil;
         move(ph, buf: ph', buf' #G:
                                  ph', buf' #F);
G:       inv len(buf) < 48 & pd≠nil & ph=nil;
         move(pd, buf: pd', buf': pd', buf' #F);
         if (len(buf) = 48) #F
         else if (active(Timer_CU)) #PART
         else #SEND
       } else receive SEND_request() {
           if (UnspecCondition) #D
           else { permit = true; #IDLE }
       }
F:     inv len(buf) = 48;
       if (permit) #B else #FULL
D:     inv ¬active(Timer_CU) or
                        UnspecCondition;
       fill(buf: buf');
```

```
B:     inv len(buf) = 48;
       send ATM_data(buf);
       set_Timer_CU;
       permit = false;
       buf = ConstructSTF (
              if (len(ph) + len(pd) > 47)  47
              else len(ph) + len(pd) );
       if (ph ≠ nil) #H
       else if (pd = nil) #IDLE
       else #G

FULL:  inv len(buf) = 48 & ¬permit;
       receive SEND_request() #B
PART: inv len(buf) < 48 & pd = nil & ph = nil;
       receive CPSpacket(pd, ATRS atrs) #C
       else receive Timer_CU()
              { if (permit) #D else #SEND }
       else receive SEND_request()
              if (UnspecCondition)
                { set Timer_CU; #D }
              else { permit = true; #PART }
SEND: inv len(buf) < 48 & pd = nil & ph = nil
             & ¬active(Timer_CU) & ¬permit;
       receive CPSpacket(pd, ATRS atrs) #C
       else receive SEND_request() #D
```

Рис. 4. Основная часть программы `Передатчик`

```
pred fill(DATA buf: DATA buf')
  pre len(buf) > 0 & len(buf) ⩽ 48
  post len(buf') = 48 & ∃pad. buf' = buf + pad & null(pad);
```

Программа `fill` расширяет список `buf` до длины в 48 октетов с обнулением добавленных октетов. Дадим оставшиеся описания.

```
bool UnspecCondition;
```

Переменная `UnspecCondition` обозначает неопределенное условие, введенное для будущей стандартизации.

```
DATA pd = nil, ph = nil, buf; bool permit;
```

Переменные `pd`, `ph`, `buf` и `permit` определяют состояние программы `Передатчик`. Флаг `permit` равен **`true`** от момента получения сообщения `message SEND_request()` до посылки очередного блока PDU.

Определим примитивы работы с таймером. Оператор `set Timer_CU` включает таймер компьютера, устанавливая его в ноль. После истечения стандартного временного интервала посылается сообщение `Timer_CU()`. Условие `active(Timer_CU)` истинно, если время еще не истекло. Всякое следующее исполнение оператора `set Timer_CU` отменяет установку предыдущего оператора.

Программа `Передатчик` имеет 11 управляющих состояний. Состояния `H` и `G` — новые. Остальные состояния — те же, что и в исходном алгоритме [2, подразд. 10.1]. Состояния `A` и `E` исходного алгоритма заменены на `G`. В коде сегмента `B` оригинального алгоритма пересылка оставшейся части заголовка пакета на буфер `buf` для случая `ph ≠ nil`, заменена переходом `#H` на вызов гиперфункции `move`, делающий тоже самое.

Реализация дополнительных требований существенно усложнила программу по сравнению с вариантом (12). Тем не менее, рассмотренная программа `Передатчик` на порядок проще по сравнению с исходной [2, подразд. 10.1]. Состояние программы определяется четырьмя переменными против семи переменных в оригинальном алгоритме (фактически девяти, однако `seq` удалена в представленном варианте при упрощении алгоритма, а переменная `tmp` — локальная). Это существенное улучшение, поскольку сложность программы экспоненциально зависит от числа переменных состояния программы.

Использование логической переменной `permit` в программе `Передатчик` — это живой пример наруше-

ния золотого правила программирования. Разумеется, можно исключить эту переменную из программы. Необходимо удалить прием сообщений `SEND_request()` в сегментах кода `IDLE` и `PART`. Модифицированная программа будет проще, однако, возможно, менее эффективной.

5. Обзор работ

Концепция автоматного программирования [1, 13] разработана Анатолием Шалыто, в том числе в интеграции с объектно-ориентированным программированием. Автоматная программа определяется в виде классического конечного автомата. Используются графическое и текстовое представления программы. Управляющие состояния являются значениями управляющей переменной, которая фактически является частью состояния программы. При реализации автоматной программы применяется switch-технология [11]. Применение объектно-ориентированной технологии позволяет "уплотнить" состояние автоматной программы, сократить ее объем и локализовать часть связей программы внутри классов. Однако спорными являются предлагаемые решения по упрятыванию управляющих состояний и самой структуры автомата внутри классов, оставляя в интерфейсе лишь объекты взаимодействия с внешним окружением.

Термин "автоматное программирование" и его аналог "*automata-based programming*" используют только в России. Тем не менее автоматные методы программирования заложены во многих языках, таких как UML, SDL [12], Дракон [14], Рефлекс [15], TLA+ [16], Event-B [17], а также LD, FBD и SFC, определяемых стандартом IEC 61131-3 для программируемых логических контроллеров. Все языки позиционируются как универсальные, однако их эффективная применимость ограничивается классом программ-процессов. Большинство этих языков являются графическими.

Иные подходы применяются в объектно-ориентированных средах программирования. В языках Java, C#, Scala [18] и других языках средства конструирования программ-процессов представлены не конструкциями языка программирования, а библиотечными классами. Их эволюция определила появление промышленных платформ, таких как BEA Oracle, Apache Geronimo, jBoss и Microsoft Windows Workflow Foundation (WF) [19]. Платформа WF является частью Microsoft .NET, определяя метасреду для конструирования, исполнения, визуализации и отладки программ-процессов в виде потоков работ[6] (*workflows*). Поток работ есть иерархически конструируемое действие (*activity*), компонентами которого являются другие действия. Стандартные действия определяются библиотечными классами Assign, Sequence, If, While, TryCatch, StateMachine, Send, Receive и др. [19, 20]. Поток работ — не объектный код, а объект среды исполнения .NET. Потери эффективности при исполнении компенсируются высокой степенью интеграции с другими сервисами .NET.

В любом языке, ориентированном на разработку программ-процессов, наряду с конструкциями для определения процессов существует базисная часть, определяющая построение обычных программ-функций. Если базисом такого языка является язык типа C++, то сложность программ-процессов усугубляется сложностью императивного программирования. Это одна из причин появления языков Erlang [21], Rust и других подобных им с функциональным языком в качестве базиса, ориентированных на быструю разработку надежных программ для телекоммуникационных, клиент-серверных, игровых, мобильных, облачных и других видов приложений, а также баз данных. Язык Erlang [21] предлагает средства построения легковесных процессов, взаимодействующих лишь посредством сообщений и не требующих синхронизации, в соответствии с концепцией акторов [22]. Состояние легковесного процесса определяется параметрами сообщений, а не переменными, поскольку в функциональном языке нет оператора присваивания.

Программу (9) моделирования работы часов с будильником можно перестроить в стиле языка Erlang. Заменим программу (9) тремя параллельными процессами `Off()`, `Set()` и `On()`, переходящими в активное состояние с помощью сообщений `of`, `set` и `on` соответственно.

```
process Off() { receive off { Часы_без_будильника() } }
process Часы_без_будильника() {
        receive H { send inc_h; Часы_без_будильника() }
        else receive M { send inc_m; Часы_без_будильника() }
        else receive A { send set }
}
```

Процессы `Set()` и `On()` определяются аналогичным образом. Сообщения `inc_h`, `inc_m`, `bell_on` и другие обрабатываются другим параллельным процессом. Его параметрами являются значения времени часов и будильника. Несмотря на большое число сообщений, эквивалентная программа на языке Erlang будет работать достаточно эффективно.

В 2009 г. архитектура легковесных процессов языка Erlang была реализована на более высоком уровне в виде библиотеки Akka [23] для языков Scala [18] и Java. Процесс в Akka может иметь состояние и его можно определить в виде конечного автомата. Пакет Akka отметился большим числом приложений; его популярность стремительно растет. Следует учитывать, что область применения Akka ограничена. Его нельзя использовать

[6] Используют также термины "бизнес-процесс" и "рабочий процесс".

для протоколов, где недопустимы потери эффективности. Например, лучшая реализация программы Передатчик для стандарта AAL-2 через пакет Akka будет проигрывать программе, представленной на рис. 4, примерно как программа (10) проигрывает программе (9), поскольку управляющие состояния конечных автоматов кодируются в Akka значениями параметров сообщений.

Заключение

Понимание программ — ключевая проблема в программировании. Эргономические методы, применяемые в графическом языке Дракон [14], существенно улучшают восприятие программы, однако не могут уменьшить ее сложность. Применение методов предикатного программирования: аппарата гиперфункций, использования рекурсивных программ вместо циклов, алгебраических типов вместо указателей и массивов существенно снижает сложность автоматных программ. Предложенные методы проиллюстрированы на программе сложного протокола AAL-2.

Большинство языков автоматного программирования являются графическими. Одна из причин этого в том, что они, преодолевая ограничения структурного программирования, обладают большей выразительностью. Тем не менее гиперграфовая структура автоматной программы, являющаяся продолжением аппарата гиперфункций, имеет более высокую степень гибкости и возможностей адекватной декомпозиции программы по сравнению с графическим представлением языка SDL [12]. С учетом новых реалий текстовое представление программы, отвечающее эргономическим требованиям, может стать основным, а графическое — вспомогательным.

Дальнейшая задача разработки технологии автоматного программирования — специализация технологии для разных подклассов программ-процессов. Сопутствующей задачей является построение классификации программ внутри класса программ-процессов. Предстоит построить иерархию подклассов и определить точные границы между ними. Большую часть класса программ-процессов составляют реактивные системы. Необходимо идентифицировать оставшуюся часть класса программ-процессов, куда входят программы, определяемые недетерминированными и вероятностными автоматами. Для класса реактивных систем следует построить внутреннюю классификацию, в частности, определить гибридные и вероятностные реактивные системы, а также временные автоматы внутри гибридных систем. В рамках работ по формальной верификации программ разработано большое число разных моделей программ. При построении классификации их надо существенно трансформировать с ориентацией на технологию программирования.

Автор благодарен А. А. Шалыто за его работы по автоматному программированию, стимулировавшие исследования автора. Автор благодарен рецензенту за множество полезных квалифицированных замечаний.

Работа выполнена при поддержке РФФИ, грант № 12-01-00686.

Список литературы

1. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование. СПб.: Питер. 2009, 176 с. URL: http://is.ifmo.ru/books/_book.pdf

2. **ITU-T Recommendation** I.363.2 (11/2000): B-ISDN ATM Adaptation Layer Specification: Type 2 AA′. URL: http://men.axenet.ru/itu/ORIGINAL/I/T-REC-I.363.2-200011-I!!PDF-E.pdf

3. **Карнаухов Н. С., Першин Д. Ю., Шелехов В. И.** Язык предикатного программирования Р. Новосибирск, 2010. 42с. (Препр. / ИСИ СО РАН; N 153).

4. **Шелехов В. И.** Разработка программы построения дерева суффиксов в технологии предикатного программирования. Новосибирск, 2004. 52 с. (Препр. / ИСИ СО РАН; N 115).

5. **Shelekhov V. I.** Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements. // Automatic Control and Computer Sciences. 2011. Vol. 45, No. 7. P. 421—427.

6. **Шелехов В. И.** Верификация и синтез эффективных программ стандартных функций в технологии предикатного программирования // Программная инженерия. 2011. № 2. С. 14—21.

7. **Шелехов В. И.** Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования. Новосибирск, 2012. 30 с. (Препр. / ИСИ СО РАН. №164).

8. **Вшивков В. А., Маркелова Т. В., Шелехов В. И**. Об алгоритмах сортировки в методе частиц в ячейках // Научный вестник НГТУ. 2008. Т. 4 (33). С. 79—94.

9. **Cooke D. E., Rushton J. N.** Taking Parnas's Principles to the Next Level: Declarative Language Design // Computer. 2009. Vol. 42, No. 9. P. 56—63.

10. **Шелехов В. И.** Предикатное программирование. Учебное пособие. Новосибирск: Изд-во НГУ, 2009. 109 с.

11. **Шалыто А. А.** SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб: Наука, 1998. URL: http://is.ifmo.ru/books/switch/1

12. **Specification** and description language (SDL). ITU-T Recommendation Z.100 (03/93). URL: http://www.itu.int/ITU-T/studygroups/com17/languages/Z100.pdf

13. **Автоматное** программирование. Статья в Википедии. URL: http://ru.wikipedia.org/wiki/Автоматное_программирование

14. **Паронджанов В. Д.** Язык ДРАКОН. Краткое описание. URL: http://drakon.su/_media/biblioteka/drakondescription.pdf

15. **Зюбин В. Е.** Программирование информационно-управляющих систем на основе конечных автоматов: Учеб.-метод. пособие. Новосибирск: Изд-во НГУ, 2006. 96 с.

16. **Lamport L.** Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers Boston: Addison-Wesley Longman Publishing Co., Inc., 2002.

17. **Abrial J.-R.** Modelling in Event-B: System and Software Engineering. Cambridge University Press, 2010.

18. **The Scala** programming language. URL:http://www.scala-lang.org/

19. **Shukla D., Schmidt B.** Essential Windows Workflow Foundation. Addison-Wesley Professional, 2006.

20. **Windows** Workflow Foundation. Microsoft Developer Network. URL: http://msdn.microsoft.com/en-us/library/dd489441.aspx

21. **Larson J.** Erlang for Concurrent Programming // ACM Queue. 2008. № 5. P. 18—23.

22. **Hewitt C., Bisshop P. and Steiger R.** A Universal Modular Actor Formalism for Artificial Intelligence // Proceedings of the 3rd International joint conference on Artificial intelligence IJCAI'73. Stanford, CA. 1973. Morgan Kaufmann Publishers Inc. San Francisco, 1973. P. 235—245.

23. **What** is Akka? URL: http://doc.akka.io/docs/akka/snapshot/intro/what-is-akka.html

УДК 004.438

Н. А. Балонин, д-р техн. наук, проф., e-mail: korbendfs@mail.ru,
М. Б. Сергеев, д-р техн. наук, проф., e-mail: mbse@mail.ru,
Санкт-Петербургский государственный университет аэрокосмического приборостроения

Реализация языка Java-MatLab в распределенных ресурсах сети Интернет

Приводится краткое описание языка математических вычислений Java-MatLab. Рассматривается связь языка, формата FB2+ и распределенных сетевых ресурсов как совокупности сетевых решений, позволяющих создать новый образ современной технической книги, содержащей исполняемые со страницы алгоритмы и наделенной связью с объектами описания.

***Ключевые слова**: язык программирования, язык программирования высокого уровня, сетевое программирование, техническая книга, сетевая робототехника, живые книги, исполняемые алгоритмы*

N. A. Balonin, M. B. Sergeev

Java-MatLab Language Release to Distributed Internet Resources

A brief description of Java-MatLab language of mathematical calculations have been done. The relationship of the language, format FB2+ and distributed network resources have been observed. The set of network solutions to create a new image of the modern technical book with pages containing executable algorithms connected to the real objects have been proposed.

***Keywords:** programming languages, high level languages, network programming, technical book, network robotics, living books, executable algorithms*

Введение

Рост возможностей сети Интернет приводит к переносу на серверы математического обеспечения, прежде используемого на персональных компьютерах. В качестве примера можно привести математическую систему "Математика", ее сетевая версия проявилась в 2009 г. на сайте [1], выдающем полноценную справку по аналитическому и численному решению дифференциальных уравнений, символьным вычислениям и многим другим разделам математики.

Одна из составляющих этой тенденции — создание формата CDF (*Computable Document Format*), способного привнести элементы математических вычислений в формат PDF [2]. Такое решение логично и исполнимо непосредственно в сети Интернет, поскольку есть два существенных фактора. Во-первых, к настоящему времени создателями сетевых ресурсов (сайтов, научных сетей) накоплен значительный опыт и необходимое программное обеспечение, а во-вторых, неотъемлемым атрибутом CDF-формата видится не только исполнение математических операций, но и связь компьютера пользователя через сеть Интернет с различными распределенными системами: сенсорами, эффекторами, учебными и научными стендами, специализированной аппаратурой и т. д. [3, 4].

В статье предлагается реализация сетевого языка математических вычислений Java-MatLab и формата FB2+, призванного расширить возможности существующей версии до использования математических вычислений в сети Интернет в режиме on-line.

Матричные операции в Java-MatLab

Язык Java-MatLab и система его реализации, разработанные авторами в рамках выполнения НИР [4] и развитые дальнейшими работами, поддерживаются на Интернет-ресурсах mathscinet.ru и livelab.ru, имеющих научную и учебную направленность [5, 6]. Оба ресурса подключены к пространственно-распределенным в сети Интернет научным и учебным стендам, к данным которых, как к сайту сети, обеспечен доступ одновре-

менно специалистов и студентов. Обработка информации со стендов ведется непосредственно в сети с использованием привычных в системе MatLab операций векторно-матричного исчисления.

Матричные операции изначально не были включены в стандарт языка Javascript. Поэтому для выделения исполняемых браузером матричных выражений была предложена конструкция с использованием двойных фигурных скобок {{...}}. Весь исполняемый в тексте Интернет-сообщения алгоритм вычисления должен размещаться между тэгами:

${{матричные операции}}$.

С помощью предкомпилятора содержимое фигурных скобок транслируется с языка векторно-матричного исчисления в Javascript. Такому исполнению авторы дали название Java-MatLab. В его реализации определены все основные матричные операции: транспонирование {{X = A'}}, алгебраические сложение {{X = A + B}}, умножение {{X = A*B}}, левое {{A = A\B}} и правое {{X = B/A}} умножение на обратную матрицу, а также поточечные операции: произведение и деление Адамара {{X = A.*B; X = A./B}}.

Выбор конструкций циклов и индексация элементов матриц с нуля, а не с единицы, как в MatLab, был выбран потому, что такой тип обозначений сейчас наиболее распространен в языковой практике. Предложение не ломать, а использовать сложившиеся привычки программистов значительно повысило эффективность использования инструмента. Возник синтетический язык, вобравший в себя оба начала: синтаксисы распространенного пакета векторно-матричного исчисления и распространенного в сети Интернет языка с синтаксисом Java. Этот синтез, резюмируя сказанное выше, и отражается в названии Java-MatLab.

В рассматриваемой реализации языка матричные формулы пишут плотно, без пробелов. Функции размещают за скобками, например, кронекерово произведение матриц выглядит как A=kron(B,C). Всего на сегодня реализовано около сотни функций [5].

С помощью Java-MatLab решают стандартные задачи линейной алгебры, включая решение систем линейных алгебраических уравнений и алгебраическую проблему собственных чисел. Этот базис позволяет развернуть процедуры анализа и синтеза линейных динамических систем, метод Рунге-Кутта, частотный анализ систем и сигналов, безошибочное решение целочисленных систем уравнений и т. п. Текущая реализация обеспечивает возможность написания и подключения пользовательских тулбоксов.

Моделирование динамических систем. По традиции язык включает в себя средства моделирования линейных динамических систем, описываемых матрицами модели пространства состояний или коэффициентами числителя N и знаменателя D передаточной функции Q(p)=N(p)/D(p). Типичный сценарий выглядит следующим образом: $t=time(10); u=one(t); N=1; D=[1,2,1]; y=lsim(N,D,u,t); plot(t,u,y);$. В данном случае оператор t=time(10) задает вектор отсчетов времени, u=one(t) генерирует входной ступенчатый сигнал, векторы N=1; D=[1,2,1] содержат числитель и знаменатель передаточной функции, y=lsim(N,D,u,t) рассчитывает выходной сигнал системы по входному, plot(t,u,y) выводит графики процессов.

Базовые вычисления линейной алгебры. Решение систем линейных алгебраических уравнений Ax=b тривиально, поскольку выполняется пользователем на уровне записи вида: $A=[[1,2,1], [2,1,2], [1,2,1]]; b=[4,5,4]; {{x=A\b}} puts('решение:'+x);$. Здесь A=[[1,2,1], [2,1,2], [1,2,1]] описывает построчно элементы матрицы системы уравнений, b=[4,5,4] задает ее правую часть. Далее следует, собственно, решение и вывод данных в строку с комментарием.

Строчную и столбцовую размерности матрицы возвращают функции n=rows(A) и m=cols(A). Это удобно для организации типичных циклов, где в качестве верхнего предела указывают n или m. На первом месте в A[i][j] стоит, как и положено, индекс строки, хотя нумерация элементов начинается с нуля.

Вещественную жорданову форму D матрицы A=VD/V и собственные векторы матрицы возвращает подпрограмма D=eig(A), V=eigv(A), и то и другое можно найти с помощью операторов M=eigs(A); D=M[0]; V=M[1]. Диагональ собственных значений выделяет D=diag(D) или D=diags(D) — при поиске комплексных величин: тогда D содержит колонки вещественных и мнимых составляющих. Вторичное применение D=diag(D) снова диагонализирует матрицу, что обычно для MatLab.

Ранг матрицы rank(A), определитель det(A), число обусловленности cond(A), разложение Холецкого chol(A), ортогонализацию по Грамму—Шмидту orth(A), QR-разложение матрицы qr(A) и многое другое можно вычислить в данной версии.

Графика и анимация. Графические возможности Интернет-языков Javascript, PHP и прочих наследуются языком Java-MatLab — все, что разработано в этой обширной теперь уже области, автоматически используется в рассматриваемом языке.

Связь языка Java-MatLab с распределенными в сети ресурсами и роботами

Математические вычисления, выполняемые в сети с использованием рассматриваемого языка, могут перетекать в создание информативных "живых" иллюстраций в "живых книгах" [4] и сетевых электронных журналах с исполняемыми алгоритмами [5, 6]. Время таких технологий пришло. Кроме того, коммутативность по отношению к внешним источникам информации, в частности, к сетевым роботам, позволяет для математических расчетов использовать данные с периферийных датчиков и влиять на эффекторы. Стоить отметить, что эта многообещающая сторона on-line использования Java-MatLab предшествующими электронными форматами книжной и журнальной продукции практически не затрагивалась.

Наиболее перспективны для расширяемого связью с роботами электронного ресурса в сети Интернет беспроводные технологии, которые испытывают настоящий бум своего развития. Сегодня область сенсорных решений в сетевых устройствах делят между собой множество реализаций с использованием стандартов передачи данных Wi-Fi, Wi-MAX, Bluetooth, Wireless USB, ZigBee, Home RF и т. д. [7]. Такие реализации легко интегрируются с сетью Интернет и представляют собой основу низкоскоростных беспроводных сетей будущего с низким энергопотреблением, предназначенных для систем управления с большим числом робототехнических узлов [7, 8].

Развитие XML-формата FB2

Эпоха семантического WEB способствовала появлению книг и документов, читаемых на букридерах и смартфонах. Немалую роль в этом сыграли появившиеся и быстро распространившиеся правила оформления документов с содержательными частями, выделенными XML-тэгами. Так это делается, например, в формате FB2.

Имеющиеся в Интернет тэги оформления таблиц вполне подходят для передачи матриц, но они избыточно сложны. Поэтому рационально по образу и подобию того, что произошло в букридерах, предложить тэги для генерации, например, портретов матриц <m>A=[[1,2], [3,4]]:опция</m>.

Анализ особенностей цветных объемных и плоских портретов матриц Адамара, Мерсенна, Эйлера и Ферма позволил развить содержательную теорию минимаксных ортогональных матриц [9—11], а также выявить неизвестные закономерности и новые артефакты. Очевидно, что с помощью таких матричных построителей можно не только передавать особенности научного исследования, но и вести его, поскольку это не просто иллюстратор, но и часть математического обеспечения, которая уже сегодня является достаточно мощной [12].

Компьютеры постепенно меняют стиль написания математических формул. Например, при помощи клавиатуры определение норм векторов в пространстве R^n проще написать так:

$$\|x\|_1 = \Sigma_{i = 1:n}|x_i|,$$

$$\|x\|_2 = (\Sigma_{i = 1:n}\, x_i^2)^{1/2},\ \|x\|_\infty = \max_{i = 1:n}|x_i|,$$

указывая границы индексов у сумм (и у интегралов, если понадобится) внизу, в строчку.

Формат, в котором реализованы указанные и многие другие возможности, развивающие и дополняющие формат FB2, авторы назвали FB2+.

Заключение

Высказанная в работах [1, 2, 12, 13] концепция построения математической сети, разумеется, не должна быть представлена отдельными (штучными) центрами развития с единственным возможным языком математической системы "Математика". Тем более, в масштабах сети Интернет. Разработчики из разных стран, математики, инженеры, творческие работники не должны быть ограничены ресурсами какой-либо одной математической империи, сколь бы крупной она ни была. Очень плодотворно на этом направлении скажется развитие хотя бы нескольких альтернативных концепций, в том числе и языка, созданного авторами предкомпилятора Java-MatLab, работающего на сайтах с математической направленностью.

Потребность в социальной сети для математиков, безусловно, назрела, и такая сеть нарастает фрагментами повсеместно, опираясь на частные разработки [13]. Но впереди у этого роста еще более грандиозная задача, поскольку в недалеком будущем громадный робототехнический комплекс будет предопределять жизнь миллионов людей. Этот комплекс будет подключен к сети в виде распределенных в пространстве роботов, и разработки взаимодействующих с ними сетевых ресурсов предваряют его появление [14]. Накопление и освещение уникального опыта в этом ключевом направлении современного развития сетевых технологий представляет собой самостоятельную и очень полезную цель, отдельные аспекты которой раскрыты в данной статье.

Список литературы

1. **Сетевой** пакет. URL: http://wolframalpha.com — последнее обращение 10.11.2013.

2. **Анонс** математической системы http://www.wolfram.com/mathematica-online с форматом CDF сохранения данных — последнее обращение 10.11.2013.

3. **Балонин Н. А., Сергеев М. Б., Балонин Ю. Н.** "Живая книга". Свидетельство о государственной регистрации программы для ЭВМ № 2012661277 от 11 декабря 2012 г.

4. **Балонин Н. А., Сергеев М. Б., Соловьев Н. В., Востриков А. А., Балонин Ю. Н., Сергеев А. М.** Отчет о НИР "Создание основ реализации дистантных систем обучения на основе технологии "Живая книга". Гос. рег. № 01201278144. СПб: ГУАП, 2013. 27 с.

5. **Математическая** сеть "Живая Книга" с Интернет-роботами и стендами. URL: http:livelab.spb.ru (основана в 2013 г.) — последнее обращение 10.11.2013.

6. **Математическая** сеть "Скайнет": технологии верстки физико-математической литературы с исполняемыми алгоритмами. URL: http:mathscinet.ru (основана в 2012 г.) — последнее обращение 10.11.2013.

7. **Балонин Н. А., Сергеев М. Б.** Беспроводные персональные сети: учебное пособие. СПб.: ГУАП, 2012. 60 с.

8. **Балонин Н. А., Сергеев М. Б.** Персональные сети WPAN на основе ZigBee. СПб.: ГУАП, 2010. 47 с.

9. **Балонин Н. А., Сергеев М. Б., Мироновский Л. А.** Вычисление матриц Адамара—Мерсенна // Информационно-управляющие системы. 2012. № 5. С. 92—93

10. **Балонин Н. А., Сергеев М. Б.** О двух способах построения матриц Адамара—Эйлера // Информационно-управляющие системы. 2013. № 1 (62). С. 7—10.

11. **Балонин Н. А., Сергеев М. Б., Мироновский Л. А.** Вычисление матриц Адамара—Ферма // Информационно-управляющие системы. 2012. № 6 (61). С. 90—93.

12. **Балонин Н. А., Сергеев М. Б.** Концепция электронного журнала с исполняемыми алгоритмами // Фундаментальные исследования. 2013. № 4—4. С. 791—795.

13. **Балонин Н. А., Сергеев М. Б.** Техническая "Живая книга": приглашение к дискуссии // Высшее образование в России. 2013. № 7. С. 141—144.

14. **Астапкович А. М., Востриков А. А., Сергеев М. Б., Чудиновский Ю. Г.** Информационно-управляющие системы на основе INTERNET // Информационно-управляющие системы. 2002. № 1. С. 12—18.

УДК 519.856.2

В. В. Федосов, канд. техн. наук, доц., г. Москва, e-mail: vlr.fdsv@gmail.com,
А. В. Федосова, канд. физ.-мат. наук, Национальный университет, г. Богота, Колумбия (Universidad Nacional de Colombia), e-mail: afedosova@unal.edu.co

Оптимизация в системе групповых выбросов–заборов загрязнений для производственной площадки (территории)

Рассмотрено загрязнение территории от группы источников выбросов и действия группы устройств забора. Алгоритм полубесконечной оптимизации определяет избыточные мощности источников выбросов, при которых выполнены нормативы загрязнения зон территории с гибкими границами. По результатам моделирования строятся карты загрязнения для сравнения вариантов развития инфраструктуры территории.

Ключевые слова: *источники выбросов, устройства заборов, компоненты выбросов, нормативы загрязняющей нагрузки территории, полубесконечная оптимизация, стохастический алгоритм, нелинейное программирование*

V. V. Fedosov, A. V. Fedosova

Optimization of the System of Group Emissions–Fences Pollution for the Production Area (Territory)

Considered contamination of the group of emission sources and the action of the fence devices. Algorithm for semi-infinite optimization determines overcapacity emission sources for which standards are made pollution zones territory with flexible boundaries. Simulation results are based pollution maps to compare options for the development of local infrastructure.

Keywords: *emission sources, device fences, components of the emission, pollution standards for areas, semi-infinite optimization, stochastic algorithm, nonlinear programming*

Введение

Вредные выбросы газообразного, капельного, пылевого типов характерны для многих производств. Если помимо источников выбросов (ИВ) в составе единой площадки (территории) размещены иные объекты или зоны (ОЗ), которые для своего функционирования требуют ограниченных (защитных) нормативов загрязнения, это может приводить к конфликту. Ослабляет конфликт дополнительное размещение устройств забора (эвакуации) загрязнений (УЗ), однако в силу неясностей с размещением и назначением мощностей таким устройствам, их влияние либо недостаточно, либо избыточно.

Если приоритетом среди размещенных участников считать нормативные ОЗ, то на общей площадке (территории) смешанная инфраструктура сможет функционировать только при ограниченных выбросах источников. Эти ограничения исходно неизвестны, так как зависят от размещения и параметров всех участников инфраструктуры.

В инфраструктуру рассматриваемой территории включены следующие группы участников: ИВ, разрешенные для реформирования; нереформируемые ИВ; УЗ; нормативные ОЗ.

В границах территории раздельно размещались точечные ИВ и УЗ с автономными симметричными выбросами (заборами), которые удобно описывать функциями с включением наиболее значимых параметров. Нормативные ОЗ представлены 2D-фигурами с гибкими границами.

Площадка или территория (далее территория) связана с масштабом задачи. Для площадки источниками выбросов являются отдельно стоящие промышленные

установки. Для территории источники выбросов — это отдельные предприятия, ТЭЦ.

Вследствие наложения автономных выбросов-заборов необходимо рассматривать загрязнение от действия всех групп источников-устройств, однако получение общего функционального описания загрязнения территории недостижимо, поэтому задачу решали численными методами.

Решение задачи устранения конфликта получено на основе алгоритмов полубесконечной оптимизации (*SIP, Semi-Infinite Programming*) [1, 2]. Моделирование и численные эксперименты выполнены в пакете MATLAB и завершены построением карт загрязнений территории. Предложенный подход совершенствует прогнозирование, контроль и управление загрязнениями территории.

Постановка задачи

Задача состоит в поиске вектора корректировки мощностей ИВ **x**, при котором функция общего загрязнения от ИВ с учетом действий УЗ, не приводит к нарушению заданных нормативов загрязнения ОЗ (**Norm**) во всех точках территории.

Условная территория **d** состоит из W зон, имеющих фиксированные границы. В зонах действуют нормативы безопасности **Norm**. В пределах территории размещены *RS* реформируемых и *NRS* не реформируемых точечных источников, а также E точечных устройств забора выбросов.

Под факелом выброса будем понимать пятно разброса частиц загрязнения с максимумом в точке выброса и понижением значений выброса к границам пятна. Факелы выбросов и заборов представлены функциями параболического типа [3], которые заменимы иными пригодными функциями, например, Гаусса [4] или практическими данными мониторинга, прогноза.

Модель представляет собой объединенное описание входных данных задачи, формулировку базового ограничения, применение алгоритма оптимизации и получения на выходе результатов в виде корректирующего вектора и графического представления загрязнений территории на начальном и конечном этапах оптимизации.

В модель введен многокомпонентный состав выбросов с числом компонентов пылевого типа $K = 7$. Каждый компонент по дисперсности δ привязан к условному диапазону размеров частиц выбросов и имеет различную дальность их разброса источниками. Оценку общего точечного загрязнения выполняли суммированием значений загрязнения от каждого активного компонента. Активным считается компонент, присутствующий в наборе компонентов выбросов частиц загрязнения данного источника выброса или (по аналогии) устройства их забора.

На рис. 1 показаны возможные связи значений и зон разброса загрязнений различного дисперсного состава выбросов при $h = -ar^2 + H$, где h — значение точечного загрязнения на удалении r от источника с условной мощностью выброса $H = 150$ у.е. [5].

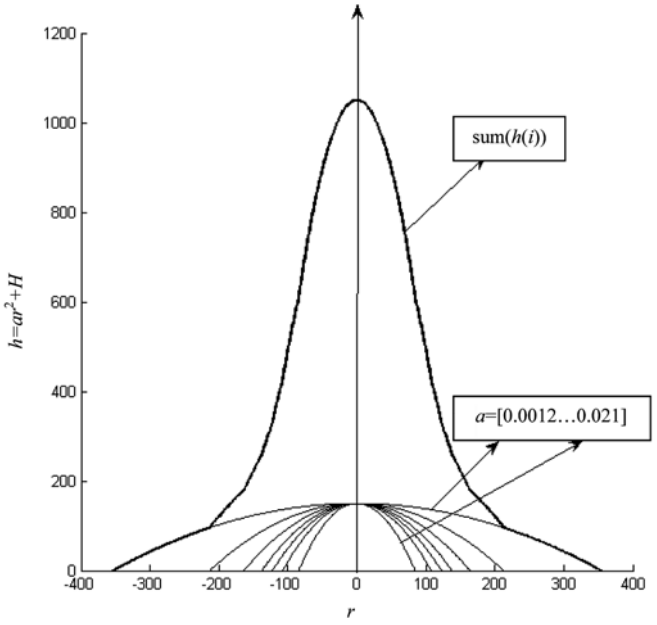

Рис. 1. Суммарное загрязнение территории одним источником в результате наложения многокомпонентного состава выбросов

Коэффициент параболы a связывается с конкретным диапазоном фракций частиц δ в составе выбросов и определяет следующее сочетание направлений векторов: $\delta\downarrow a\downarrow R\uparrow$ (R — максимальный радиус разброса частиц). Выделенной линией показано распределение суммарных загрязнений от источника. Такое распределение соответствует полному набору компонентов в составе выбросов ($K = 7$). Переменный состав выбросов источников учитывался введением в модель матрицы (наборов) компонентов [1, 0]. Существование наборов компонентов приводит к вариациям суммирующего распределения загрязнения по каждому источнику.

Основные переменные модели:

- расположение и конфигурация ОЗ территории (*карта*);
- мощности реформируемых источников (**H**), не реформируемых источников (**NH**) и устройств забора (**F**);
- координаты реформируемых источников (**tN**), не реформируемых источников (**tNN**) и устройств забора (**tE**);
- наборы компонентов выбросов реформируемых источников (**kod1**), не реформируемых источников (**kod2**) и устройств забора (**kod3**);
- коэффициенты разброса компонентов реформируемыми источниками (**a1**), не реформируемыми источниками (**a2**) и устройствами забора (**a3**);

В любой точке s территории декларировано соблюдение ограничения

$$g(s) = \sum_{j=1}^{RS}\sum_{k=1}^{K} kod1(k,j)(-a1(k)r(s,j)^2 + (1 - x(j))H(j)) +$$
$$+ \sum_{j=1}^{NRS}\sum_{k=1}^{K} kod2(k,j)(-a2(k)r(s,j)^2 + NH(j)) -$$
$$- \sum_{j=1}^{E}\sum_{k=1}^{K} kod3(k,j)(-a3(k)r(s,j)^2 + F(j)) - Norm(\omega(s)) \leqslant 0, \quad (1)$$

где $x(j)$ — коэффициент реформирования мощности источника j; $r(s,j)$ — расстояние от s до источника (устройства) j; k — номер компонента; $\omega(s)$ — номер ОЗ по карте размещения; $Norm(\omega(s))$ — норматив загрязнения для точки s зоны $\omega(s)$.

Ограничение (1) сопоставляет разницу между уровнями выбросов частиц загрязнения от источников и уровнями их забора от устройств с заявленным нормативом в каждой точке территории. В частности, сумма выбросов всех реформируемых источников по всем активным компонентам (строка 1) плюс сумма выбросов всех нереформируемых источников по всем активным компонентам (строка 2) за вычетом суммы заборов всех устройств по всем активным компонентам (строка 3) не должна превышать заданный норматив загрязнений в любой точке ОЗ территории (конец строки 3).

В первом слагаемом ограничения (строка 1) присутствует $(1 - x(j))$, что указывает на группу реформируемых источников. На старте $x(j) = 0$, после оптимизации $0 \leqslant x(j) \leqslant 1$. Таким образом, мощности некоторых источников будут реформированы, что определит разрешение конфликта и приведет к оптимальной карте общего загрязнения территории. Ценой разрешения конфликта будут потери от снижения выпуска условной продукции при сокращении мощностей источников выбросов.

Алгоритм решения задачи

Теоретические положения и обоснование алгоритмов SIP, которые использовались в настоящей работе, имеют обширную библиографию. В приложениях алгоритмов указаны полевые задачи, сходные с проблемами промышленной экологии.

Полубесконечность рассматриваемой задачи вытекает из конечного числа реформируемых источников, но бесконечного числа ограничений, так как число точек территории в силу неопределенной дискретизации бесконечно. Если перейти к конечной задаче по источникам и ограничениям, то она решается стандартными алгоритмами линейного или нелинейного программирования.

Алгоритмы SIP находят из множества невыполнимых ограничений ограничения критические, что позволяет перейти к конечной задаче.

Система MATLAB особенно удобна для таких задач, так как ориентирована на операции с матрицами, содержит готовые функции стохастического выбора, максимизации параметра по нескольким переменным, линейного и нелинейного программирования, а также мощную и быструю графику.

Согласно алгоритму на *первом* этапе стохастической процедурой в пределах объявленного лимита попыток обнаруживают точки невыполнения ограничения (1). На *втором* этапе ограничение (1) рассматривают в качестве функции координат территории и выполняют ее максимизацию от точек старта первого этапа. Максимизация обнаруживает координаты *smax* ближайшего локального максимума невыполнения ограничения ЛМН (1). На *третьем* этапе, если ЛМН признан новым и в нем также не выполняется ограничение (1), по его параметрам рассчитывают строку глобальной матрицы оптимизации. Появление новой строки матрицы позволяет заново решить задачу оптимизации, хотя итерации можно проводить для *пакета* новых строк. Пакет набирается из строк, обнаруженных в пределах лимита попыток. Решением является вектор изменения мощностей источников **x**. Завершением алгоритма могут быть: прекращение изменения целевой функции, отсутствие новых точек невыполнения ограничения (1) в пределах лимита попыток, стабилизация **x**.

Стартовые точки невыполнения ограничения (1) всегда находятся в области **d**. Координаты обнаруженных ЛМН *smax* могут располагаться как в **d**, так и за пределами **d** (с сохранением или со сменой знака координат).

Коэффициенты строки и правый член матрицы оптимизации для обнаруженного нового ЛМН рассчитывают по следующим формулам:

$$A(j{:}RS) = \sum_{j=1}^{RS}\sum_{k=1}^{K} \begin{cases} -H(j), & \text{если } r(s,j) < R(k,j) \quad \text{и} \quad kod1(k,j) = 1 \\ 0, & \text{если } r(s,j) \geqslant R(k,j) \text{ или } kod1(k,j) = 0, \end{cases} \quad (2)$$

$$b = Norm(\omega(s)) - \sum_{j=1}^{RS}\sum_{k=1}^{K} kod1(k,j)(-a1(k)r(s,j)^2 + H(j)) - \sum_{j=1}^{NRS}\sum_{k=1}^{K} kod2(k,j)(-a2(k)r(s,j)^2 + NH(j)) +$$
$$+ \sum_{j=1}^{E}\sum_{k=1}^{K} kod3(k,j)(-a3(k)r(s,j)^2 + F(j)). \quad (3)$$

В матрице оптимизации присутствуют (> 0) лишь те источники выбросов, которые активны к выявленному ЛМН. Условия активности: $r(s, j) < R(k, j)$, где $R(k, j)$ — радиус действия источника j для компонента k, и не нулевое присутствие компонента k в наборе данного источника j. Это означает, что источник в отношении случайной точки s считается активным, если хотя бы один из его набора компонентов при разбросе частиц загрязнения достигает этой точки.

Целевая функция $f(x)$ ориентирована на стоимостную оценку снижения интегральных выбросов источников в результате решения задачи. Интегральные выбросы вычислялись суммированием объемов параболоидов (см. рис. 1). Таким образом:

$$\min\left\{f(x) = \pi \sum_{j=1}^{RS} \sum_{k=1}^{K} \frac{kod1(k,j)mp(j)x(j)^2 H(j)^2}{2a1(k)}\right\}, \quad (4)$$

где $mp(j)$ — единичные стоимости потерь от понижения интегральной мощности источника j.

Отметим, что допустимы более развернутые формулировки целевых функций с усложненным технико-экономическим смыслом.

Численные эксперименты

На рис. 2 (см. третью сторону обложки) показана карта размещения на территории объявленных зон ($W = 6$) с разными нормативами безопасности. Допустимо встраивание "свободных" зон, для которых требования безопасности превышают суммарные выбросы.

В зонах указаны их нормативы безопасности. Наиболее жесткий норматив ОЗ 1:150 имеют две зоны, а наиболее ослабленный (ОЗ 5:572) одна.

Рис. 2 также содержит карту размещения групп реформируемых ИВ (красные), не реформируемых ИВ (синие) и группы УЗ (зеленые). Возле точек размещения указаны их мощности (у. е.).

В табл. 1 представлены исходные данные реформируемых источников ($RS = 11$) в экспериментах: мощности источников; координаты их размещения; наборы компонентов.

В табл. 2 совмещены исходные данные не реформируемых источников ($NRS = 3$) и устройств забора ($E = 8$) в экспериментах, а именно: мощности; координаты их размещения; наборы компонентов.

В табл. 3 приведены векторы коэффициентов парабол разброса (забора) загрязнений для обеих групп ИВ и группы УЗ. Интенсивности разброса в группе не реформируемых источников выше, чем в группе реформируемых источников. Интенсивности разброса группы устройств забора определяет самостоятельный вектор.

В эксперименте лимиты стохастического поиска на итерации ограничивались 500. Этого достаточно для эффективного продвижения к оптимальному решению (4...6 итераций).

В табл. 4 представлено найденное оптимальное решение **x** — вектор понижения мощностей источников (относительно исходного **H**, табл. 1). В решении источники 3 и 8 сохранили исходную мощность выбросов; источники 4 и 5 должны быть выведены из рассмотрения, как неприемлемые по вкладу в общие выбросы, а остальные источники нуждаются в реформировании. Общая относительная мощность источников выбросов в результате понижения составит $\Sigma(1 - \mathbf{x})\mathbf{H}/\Sigma\mathbf{H} = 0{,}477$.

Покажем роль найденного решения в графике эксперимента.

Таблица 1

Реформируемые источники

Параметры	Источники										
	1	2	3	4	5	6	7	8	9	10	11
H	426	362	51	365	253	951	587	619	411	286	423
tN	776	388	114	733	768	29	547	606	314	137	25
	383	320	169	317	262	340	87	297	262	282	111
kod1	1	0	1	1	0	1	0	0	1	1	1
	1	1	0	0	0	0	1	0	1	0	1
	0	1	1	1	0	0	0	1	0	1	1
	1	0	0	0	1	0	1	1	1	0	0
	0	1	0	1	0	0	0	0	1	1	1
	0	1	1	0	1	1	0	0	0	0	0
	0	0	1	1	0	1	0	1	0	0	1

Таблица 2

Не реформируемые источники и устройства забора

Параметры	Источники			Параметры	Устройства							
	1	2	3		1	2	3	4	5	6	7	8
NH	216	98	194	**F**	552	268	576	486	222	666	624	308
tNN	193	704	500	**tE**	45	636	291	517	604	544	130	399
	62	241	349		199	75	178	284	110	262	48	384
kod2	0	1	0	**kod3**	1	1	1	1	0	1	0	0
	1	1	0		1	1	0	1	1	1	1	1
	0	0	1		1	0	0	1	0	0	0	1
	1	0	1		1	1	0	0	1	0	0	1
	1	0	1		0	1	1	1	0	1	1	1
	1	1	0		1	1	0	0	1	0	1	0
	0	1	0		0	1	1	0	0	1	0	1

Таблица 3

Коэффициенты парабол разброса компонентов ($\times 10^{-2}$)

Параметры	Компоненты						
	1	2	3	4	5	6	7
a1	0,66	1,82	3,08	4,35	5,50	7,15	11,55
a2	0,42	1,16	1,96	2,77	3,50	4,55	7,35
a3	1,04	1,47	2,05	2,54	2,85	3,44	3,83

Таблица 4

Вектор оптимального решения

Решение	Источники										
	1	2	3	4	5	6	7	8	9	10	11
x	0,85	0,53	0	1,00	1,00	0,60	0,42	0	0,44	0,61	0,30

Сравнительная графика

На рис. 3 (см. третью сторону обложки) показана исходная карта общего загрязнения территории выбросами, построенная на основе данных табл. 1, 2. Наиболее загрязнена левая часть территории, а также ее северо-восточная область. Размещения и порядки возвышений уровней загрязнений значительно больше объявленных нормативов ОЗ, что приводит к необходимости оптимизации. Рассмотрим отдельные положения, трактующие (интерпретирующие) содержание карты.

Возвышение [$X \approx 0...65$, $Y \approx 100...170$] в значительной степени сформировано выбросами источников 11 и 3. У источника 11 достаточно высокая мощность (423 у.е.) и активны пять компонентов выбросов из семи. Все они при этом работают на примыкающую к источнику область. Поддержка выбросами маломощного источника 3 (мощность 51 у.е.) также имеет место, поскольку активны четыре компонента.

Напротив, вблизи наиболее мощного в группе источника 6 (951 у.е.) загрязнения менее выражены, что объясняют активности только трех компонентов. В еще меньшей степени зафиксировано картой загрязнение вблизи достаточно мощного источника 7 (587 у.е.), но с активностью только двух компонентов.

Ввод в модель векторов **kp1** и **kp2** регулирует дальность разброса компонентов. Следовательно, не для всех точек территории активности компонентов источника означают присутствие их выбросов на карте.

Таким образом, логика карты продиктована наложением выбросов источников с объявленными параметрами мощности, активности компонентов и дальности их разброса.

На рис. 4 (см. третью сторону обложки) показаны результаты работы группы устройств забора загрязнений. При меньшей в сравнении с источниками численности группы, меньшей суммарной мощности и меньших радиусах разброса эффективность устройств достаточно высока и практически охватывает всю территорию. Например, относительная активность группы устройств по компонентам составляет 33/56 против обеих групп источников 48/98.

На рис. 5 (см. четвертую сторону обложки) показана карта остаточного загрязнения территории в результате действия всех ИВ и УЗ модели. Благодаря мощностям, активности и размещению УЗ более половины территории свободно от загрязнений. Однако остаются значительные области загрязнения с собственными неодинаковыми возвышениями. Во многих точках территории нормативы ОЗ явно превышены и следовательно, задача нахождения понижающего вектора **x** ИВ остается актуальной.

С учетом найденного понижающего вектора (см. табл. 4) меняется карта оптимального загрязнения территории от ИВ (рис. 6, см. четвертую сторону обложки).

В частности, значительно ослаблены выраженные возвышения в левой части территории (см. рис. 3). Пик на северо-востоке также изменил конфигурацию: уменьшились уровни загрязнения, а новое возвышение сдвинуто и сформировалось из бывшего склона. По наличию полностью свободных областей карты территории до оптимизации (см. рис. 3) и после оптимизации (см. рис. 6, см. четвертую сторону обложки) мало изменились.

Наложение не изменившейся карты действия УЗ (см. рис. 4) на карту оптимальных ИВ (см. рис. 6) показано на карте остаточного загрязнения территории после оптимизации (рис. 7).

Практически вся территория свободна от загрязнения и удовлетворяет назначенным нормативам ОЗ. Обращают на себя внимание два фактора: остатки расположены только в приграничных областях территории; нормативы ОЗ нарушены только в части точек областей остатков. Например, в области остатков на северо-востоке допустим уровень загрязнения ОЗ 6:455; для зоны на северо-западе — ОЗ 4:488, ОЗ 5:572. Небольшая зона остатков на юге также уменьшится с учетом разрешенного норматива ОЗ 2:221.

Более конкретно наличие остаточных загрязнений объясняют особенности алгоритма задачи и взаимодействие составляющих элементов ограничения (1). Например, максимизация функции ограничения с точек старта вблизи границ территории может обнаружить новые ЛМН за пределами объявленных границ **d**. Алгоритм не рассматривает ЛМН, покинувшие территорию (их признают виртуальными). Глобальная матрица, таким образом, не содержит строк, рассчитанных по их параметрам, и генерирует решение без их участия. Или не исключены ситуации для точек, при которых априори независимо от вкладов реформируемых источников ограничение будет постоянно нарушенным:

$$\Sigma\mathbf{kod2}(-\mathbf{a2}r^2 + \mathbf{NH}) - \\ - \Sigma\mathbf{kod3}(-\mathbf{a3}r^2 + \mathbf{F}) - \mathbf{Norm} > 0.$$

Если наличие остатков загрязнений неприемлемо, в распоряжении экспериментатора широкие возможности корректировки стартовых параметров участников моделирования либо просто изменения координат их привязки.

Некоторые из объявленных входящих параметров допускают более углубленную трактовку. Так, матрицы $\mathbf{kod1}_{K \times RS}$, $\mathbf{kod2}_{K \times NRS}$, $\mathbf{kod3}_{K \times E}$ формируют наборы (активности) компонентов ИВ или УЗ. Это позволяет быстро оценить результаты оптимизации в зависимости от содержания компонентов в выбросах или заборах. Так в численном эксперименте у 11 реформируемых источников будет 37 активных компонентов (число единиц в **kod1**, см. табл. 1), у 3 не реформируемых источников — 11 (число единиц в **kod2**, см. табл. 2), у 8 устройств забора — 33 (число единиц в **kod3**, см. табл. 2). В результате 14 источникам выбросов с $\Sigma\mathbf{kod1} + \Sigma\mathbf{kod2} = 37 + 11 = 48$ активными компонентами противостоят 8 устройств забора с $\Sigma\mathbf{kod3} = 33$ активными компонентами.

При переходе к интервальным значениям ($0 < m < 1$) элементов матриц **kod1** и **kod2** достигается быстрое избирательное варьирование мощностью выброса или забора по любому из компонентов с просмотром результатов в графике. Кроме того, присвоение долевого значения элементу матрицы можно отождествить с наличием фильтра у данного источника выброса по данному компоненту. Например, если $kod1(4, 5) = 0{,}85$, то этот факт означает, что источник 5 по компоненту 4 оборудован избирательным фильтром, задерживающим 15 % выброса.

Оптимизация показывает цену решения поставленной задачи: соблюдение заявленных нормативов ОЗ территории требует понижения потенциала выбросов территории на $1 - 0{,}477 = 0{,}523$ (исходный потенциал принят за 1). Будут ли компенсированы такие потери за счет роста экологического статуса территории или учета иных показателей и соображений зависит от лиц, принимающих решения. В любом случае

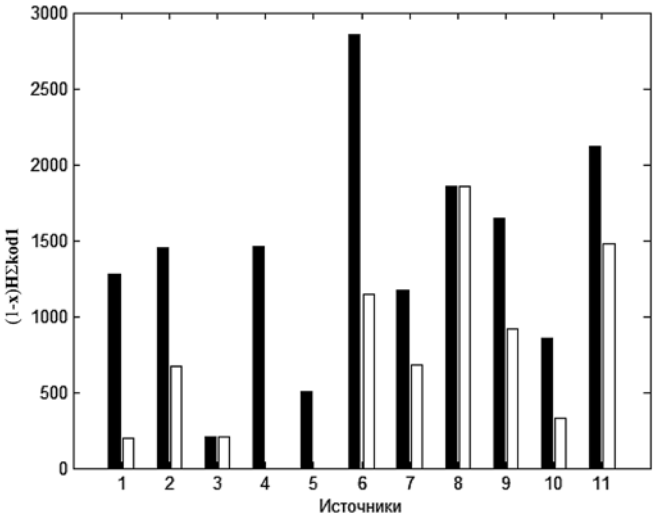

Рис. 8. Диаграмма критериев активности реформируемых источников до и после оптимизации

остается возможность быстрого и неоднократного переформатирования вводных параметров модели для достижения приемлемого результата.

Появляется возможность формулирования наглядных критериев сопоставления как результатов оптимизации, так и иных вариантов карт размещения участников. На рис. 8 приведены оценки источников реформируемых выбросов до (темные) и после (светлые) оптимизации по критерию, обобщающему мощности и активности компонентов выбросов. Вектор изменившихся критериев продиктован полученным оптимальным решением. Из трех исходно самых сильных источников только источник 8 полностью сохранил свой потенциал, а источники 6 и 11 лишь частично. Источники 4 и 5 утратили его полностью. Заметное и различное падение промышленного потенциала источников в дальнейшем может быть увязано, например, с сокращением производимых полезных продуктов, изменением их себестоимости или т. п.

Графика результатов моделирования позволяет строить и анализировать не только общие, но также покомпонентные карты загрязнений территорий.

Модель содержит достаточно ясный постановочный набор параметров, не содержит трудно расшифровываемых коэффициентов, открыта для дополнений (например, за счет добавления ограничений, типа и числа участников и т. п.), легко переходит на иные целевые функции. Она может быть применена при выборе или анализе приоритетов развития территории в условиях конфликта избыточных загрязнений с требованиями экологии.

Заключение

Рассмотрен конфликт размещения в пределах площадки (территории) групп источников выбросов, эвакуационных устройств и объектов (зон) с защитными нормативами загрязнения. Выбросы имеют многокомпонентную структуру.

На основе алгоритмов полубесконечной оптимизации (SIP) определен вектор корректировки мощностей источников выбросов, при котором возможно функционирование совместной инфраструктуры территории.

Программный комплекс строит карты загрязнения на всех этапах оптимизации и может служить инструментом оперативного сравнения различных вариантов проектирования или развития промышленных площадок (территорий).

Список литературы

1. **Volkov Y. V. and Zavriev S. K.** A general Stochastic Outer Approximations Method // SIAM Journal of Control and Optimization. 1997. N 35 (4). P. 1387—1421.

2. **Завриев С. К., Новикова Н. М., Федосова А. В.** Стохастический алгоритм решения выпуклых задач полубесконечной оптимизации с ограничениями равенствами и неравенствами // Вестник Московского университета. Сер. 15. Вычислительная математика и кибернетика. 2000. N 4. С. 30—35.

3. **Федосов В. В., Федосова А. В.** Полубесконечная модель фильтров многокомпонентных выбросов для группы промышленных источников территории // Безопасность жизнедеятельности. 2013. № 2. С. 42—46.

4. Vaz A. I. F., Ferreira E. C. Air pollution control with semi-infinite programming // Applied Mathematical Modelling. 2009. N 33. P. 1957—1969.

5. Федосов В. В., Федосова А. В. Стохастический поиск защитных нормативов объектов (зон), размещенных на общей территории с группой источников промышленных выбросов // Информационные технологии. 2013. № 11. С. 27—32.

УДК 004.423

Р. Ю. Замараев, канд. техн. наук, ст. науч. сотр., **С. Е. Попов,** канд. техн. наук, ст. науч. сотр.,
Федеральное государственное бюджетное учреждение науки Институт вычислительных технологий Сибирского отделения Российской академии наук, г. Кемерово,
e-mail: popov@ict.sbras.ru

Программный комплекс для классификации региональных сейсмических событий "Сейсматика"

Рассмотрен программный комплекс "Сейсматика", предназначенный для классификации сейсмических событий. Программный комплекс построен на базе математических моделей состояния сейсмических генераторов, которые трактуются как характеристические функции "волновода" из очага к сейсмостанции. "Информационным ядром" комплекса является база данных сейсмических шаблонов, которые представлены как качественные записи ударных волн, приуроченные к событиям с известным генезисом, имеющие определенный набор характеристик.

Ключевые слова: *программный комплекс, веб-сервис, классификация сейсмических событий функциональные показатели, анализ данных*

R. Y. Zamaraev, S. E. Popov

The Software Package "Seismatica" for Classification of Seismic Events

The article describes the software package "Seismatica" intended for classification of seismic events. Software package is based on mathematical models of state seismic generators, which are treated as characteristic functions "waveguide" from the source to the seismic station. "Information core" of the complex is a database of seismic patterns that are presented as high-quality recordings of shock waves, confined to events with known genesis, having a certain set of characteristics.

Keywords: *software package, web service, classification of seismic events, functional indicators, data analysis*

Введение

Мониторинг и анализ региональной геодинамической ситуации в Кузбассе отличаются высоким уровнем ответственности и сложности решаемых задач. Причина в том, что наряду с мощными возмущениями из известных очаговых зон, анализировать и классифицировать приходится разнородный поток событий, среди которых промышленные взрывы различной мощности и глубины заложения, горные удары и оползни. Важным аспектом такого анализа является также подтверждение (или опровержение) связи местных и региональных сейсмических событий с техногенным воздействием на геосреду.

На настоящее время существует ряд специализированных систем, предоставляющих функциональные возможности для анализа сейсмической информации. К их числу относятся, например, программный комплекс GeoSeisQC [1], предназначенный для контроля за производством сейсморазведочных работ и оценки качества сейсмических данных. С помощью этого комплекса удается получить оценки качества сигналов по частотному и амплитудному составам, значения атрибутов сейсмических записей во временном окне и т. п. Другая система Stratimagic с модулем NexModel [2] предназначена для интерактивного моделирования сейсмического волнового поля. Она позволяет провести количественную проверку геологических гипотез и поэтапную классификацию сейсмических данных.

Важной в плане решения основной задачи является организация доступа к большим массивам актуальных сейсмологических данных, а главное, их оперативная обработка и анализ с использованием новых (или модифицируемых) математических моделей. На настоящее время существует ряд продвинутых в этом

направлении инструментальных средств и систем. К их числу относится, например, прототип веб-службы для решения обратной геофизической задачи — совместный проект Института физики земли, Института системного анализа и Всероссийского института научной и технической информации РАН. Эта веб-служба использует территориально-распределенную грид-инфраструктуру, построенную на базе программного комплекса Globus Toolkit [3].

Однако функциональные возможности существующих программных решений не реализует полностью потребности рассматриваемой предметной области. Они не поддерживают интеграцию онлайн-сервисов с системой управления базами данных сейсмической информации и с инструментальными средствами создания, тестирования и выполнения программных компонентов анализа сигналов и классификации событий.

В связи с изложенными выше соображениями актуальной представляется разработка регионально-ориентированных, эргономичных сервисов для классификации сейсмических событий, оперативной оценки причинно-следственных связей и контроля соблюдения правил ведения горных работ.

О методах классификации сейсмических событий

Обратные задачи сейсмологии традиционно являются и остаются на сегодняшний день практически значимыми и актуальными. Общими для них были и остаются вопросы идентификации параметров динамических и статистических моделей очага, которые создает априори существующая нестационарность сейсмических процессов. Она проявляется как в самом факте появления высокоэнергетического события, так и в структуре сейсмологических сигналов, соответствующих этому событию.

В области классификации сейсмических событий существует ряд представляющих интерес исследований и результатов. В работе [4], например, рассмотрены изменения средних значений временных интервалов между сейсмическими событиями и их коэффициент вариации. На основе представлений о прочности твердых тел и базирующейся на них иерархической модели разрушения горных пород сформулированы физически обоснованные критерии формирования очаговой стадии процесса и построен прогноз места, времени и энергии сейсмических явлений. В работе [5] рассмотрены основные классы событий, к которым авторы относят вулкано-тектонические, региональные, долгопериодные, гибридные события, а также камнепады. Идентификация и классификация этих процессов происходит в искусственных нейронных сетях на основе паттернов сигналов (или их преобразований) для событий известных классов.

Представленный в настоящей статье программный комплекс опирается на диагностический подход к наблюдениям и экспериментам. В основе такого подхода лежит энтропийная модель выборочных данных, позволяющая использовать форму сейсмологического сигнала как классификационный признак. С одной стороны, это позволяет решать задачу классификации сейсмических событий на основе узнаваемых и структурированных особенностей. С другой стороны, при этом сохраняется привязка наблюдаемых отличий сигналов к временной области, а следовательно — возможность получения из теории колебаний правил интерпретаций событий.

Математический алгоритм классификации сейсмических событий

В программном комплексе реализован энтропийный метод анализа нестационарных процессов (ЭМАНП), описание и обоснование которого можно найти в работах [6—8].

Согласно ЭМАНП матрица исходных данных $\{x_{ij}\}$, $i = 1, ..., m, j = 1, ..., n$ заменяется матрицей размахов

$$d_{ij} = (x_{ij} - x_{i+1j})^2,$$

обеспечивающей строгую аддитивность элементов. Такая замена необходима для вычисления матрицы весов

$$q_{ij} = \frac{d_{ij}}{\sum_{i=1}^{m} d_{ij}}$$

и затем матрицы энтропий

$$E_{ij} = -q_{ij}\ln(q_{ij}).$$

В матрице энтропий обеспечивается аддитивность не только элементов, относящихся к одному сигналу (вектору-столбцу), но и аддитивность элементов из различных сигналов, так как матрица E соответствует определению мультиканального источника сообщений из теории информации. Отсюда может быть вычислена следующая матрица, как обобщение (сумма) информации о сигналах по трем каналам измерений:

$$H_{ik} = E_{ij+1} + E_{ij+2} + E_{ij+3},$$

где $k = 1...n/3$, $j = 3(k - 1)$. Подавление шумов и получение прогнозируемых по форме характеристических функций событий осуществляется аккумулированием значений в столбцах матрицы H, дает матрицу

$$C_{ik} = \sum_{l=1}^{i} H_{lk}.$$

Согласно ЭМАНП для получения матрицы информативности по Байесу, назовем ее D, необходимо провести следующую процедуру стандартизации:

$$D_{ik} = \frac{D_{ik} - \mu_i}{\sigma_i},$$

где $\mu_i = \frac{3}{n} \sum_{k=1}^{n/3} C_{ik}$ и $\sigma_i = \sqrt{\frac{3}{n} \sum_{k=1}^{n/3} (C_{ik} - \mu_i)}$.

Теперь подобие двух событий как двух диагнозов, представленных векторами-столбцами с номерами u и v, может быть оценено любым удобным способом. В программном комплексе подобие классифицируемого события каждому известному событию в наборе оценивается по коэффициенту корреляции Пирсона.

Исходные данные

Программный комплекс использует специально разработанную базу данных региональных сейсмических событий (рис. 1). Она содержит шаблоны событий, а именно фрагменты суточных записей (рис. 2), приуроченных к выдающимся по энергии событиям с набором следующих атрибутов: время, магнитуда, класс, локализация, экспертная классификация и др. База данных комплекса функционирует на основе СУБД PostgreSQL и в настоящее время насчитывает более 45 тыс. уникальных записей о сейсмических событиях, произошедших на территории Кемеровской области и прилегающих к ней регионов.

Первичным источником данных является региональная сеть из восьми сейсмических станций (рис. 3) различной ведомственной принадлежности со следующими международными кодами: ASR1, ELT, BRCR, KEM, LUZB, NVS, SALR, TASR.

Каждая станция ведет непрерывную запись сейсмических колебаний с частотой дискретизации 100 Гц. Сигналы поступают по трем ортогональным в пространстве каналам в формате miniSEED. Длина шаблона опытным путем принята (избыточно) равной 18000 отсчетам. Начало шаблона приравнено апостериорному времени события с учетом задержки прихода сейсмической волны из идентифицированного географически очага, что обеспечивает захват основного пика колебаний (см. рис. 2).

daynumber	Номер дня
date	Дата события
id	Номер события
type	Тип землетрясения
gmt	Время по Гринвичу
lnglat	Географическая привязка
class	Класс
magnitude	Магнитуда
observatory	Пункт наблюдения
nearestsettlement	Ближайший населенный пункт
objectname	Место события
explosiveamount	Количество взрывчатки
starttime	Начальное время шаблона
length	Длина шаблона
samplerate	Частота дискретизации
data	Значения сигнала
comments	Комментарии
uid	Уникальный идентификатор шаблона
channels	Список каналов
networkcode	Код сети
stationid	Идентификатор станции
locationid	Идентификатор потока данных

Рис. 1. Таблица шаблонов сейсмических событий

Для устранения неинформативных шумов на этапе подготовки данных выполняется обрезка шаблона справа и слева. Длина обрезки варьируется от удаленности рассматриваемого участка от выбранной сейсмостанции, но общая длина всех шаблонов, передаваемых на вход алгоритму, должна быть одинаковой.

Данные, передаваемые на вход программе, реализующей математический алгоритм классификации (см. ниже), заданы набором шаблонов сейсмических событий. Их пример представлен на рис. 4.

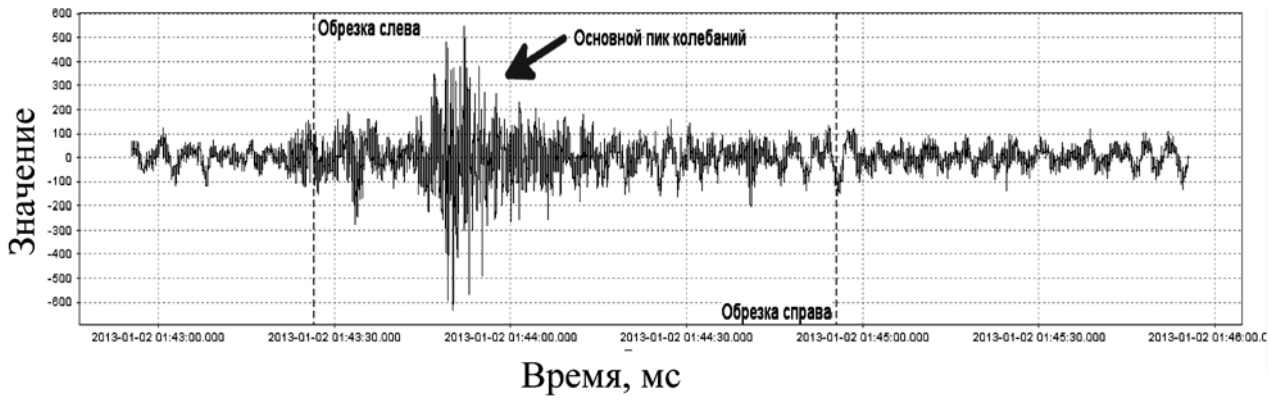

Рис. 2. Пример сейсмологического сигнала

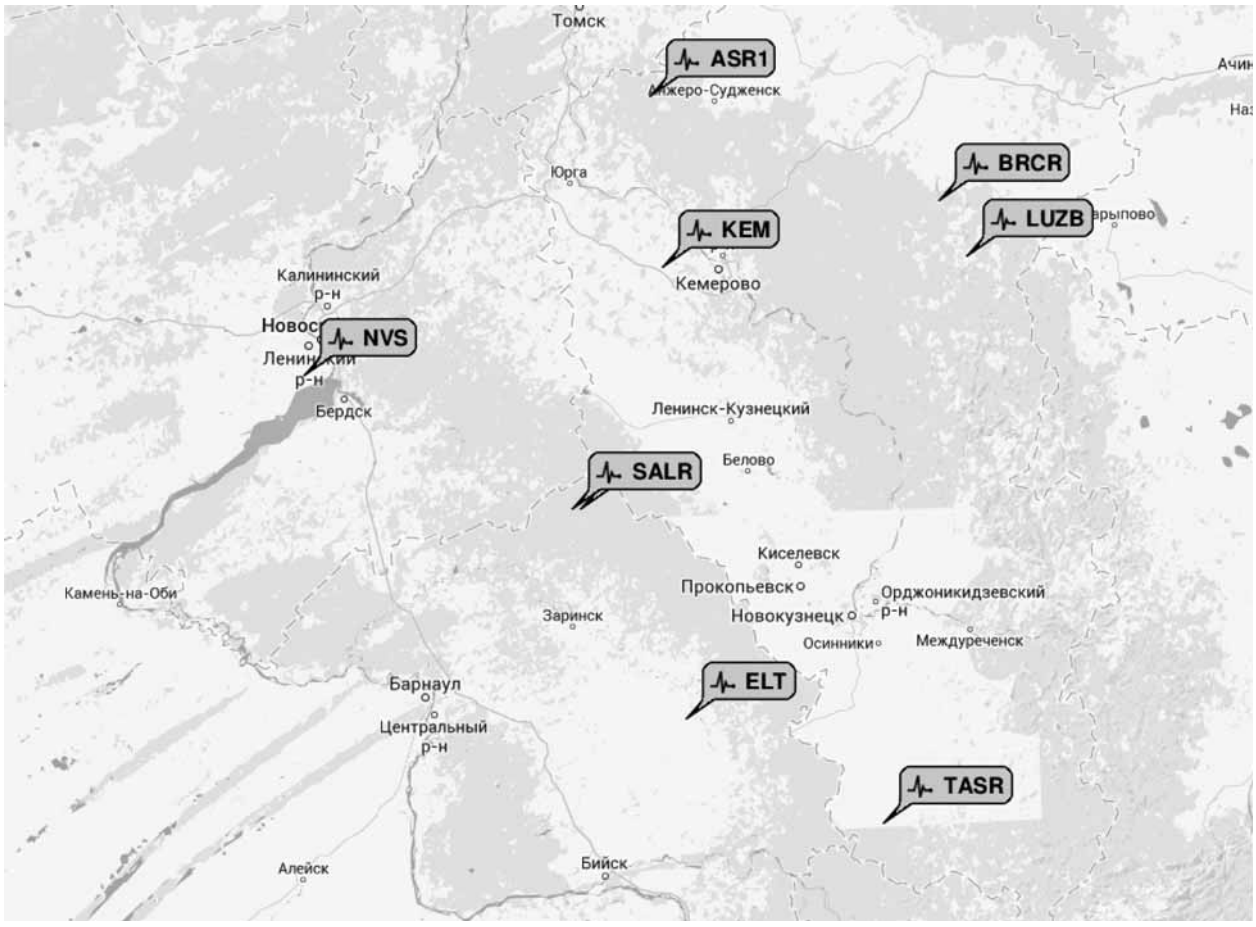

Рис. 3. Географическое расположение сейсмостанций

	Классифицируемое событие			Шаблоны						
	EHE	EHN	EHZ	EHE	EHN	EHZ	.....	EHE	EHN	EHZ
m	-192	45	266	142	278	-50		-234	140	266
	-112	-523	250	142	-248	-255		352	903	737
	158	147	354	302	246	456		027	342	290
	⋮	⋮	⋮	⋮	⋮	⋮	....	⋮	⋮	⋮
	1320	-389	373	686	1632	1396		-308	-308	1528
	1567	-401	122	543	1200	1195		-248	-673	2698
	n									

Рис. 4. Структура входных данных:
m — число отсчетов в сигнале; *n* — общее число каналов; EHE, EHN, EHZ — идентификаторы каналов сейсмостанции

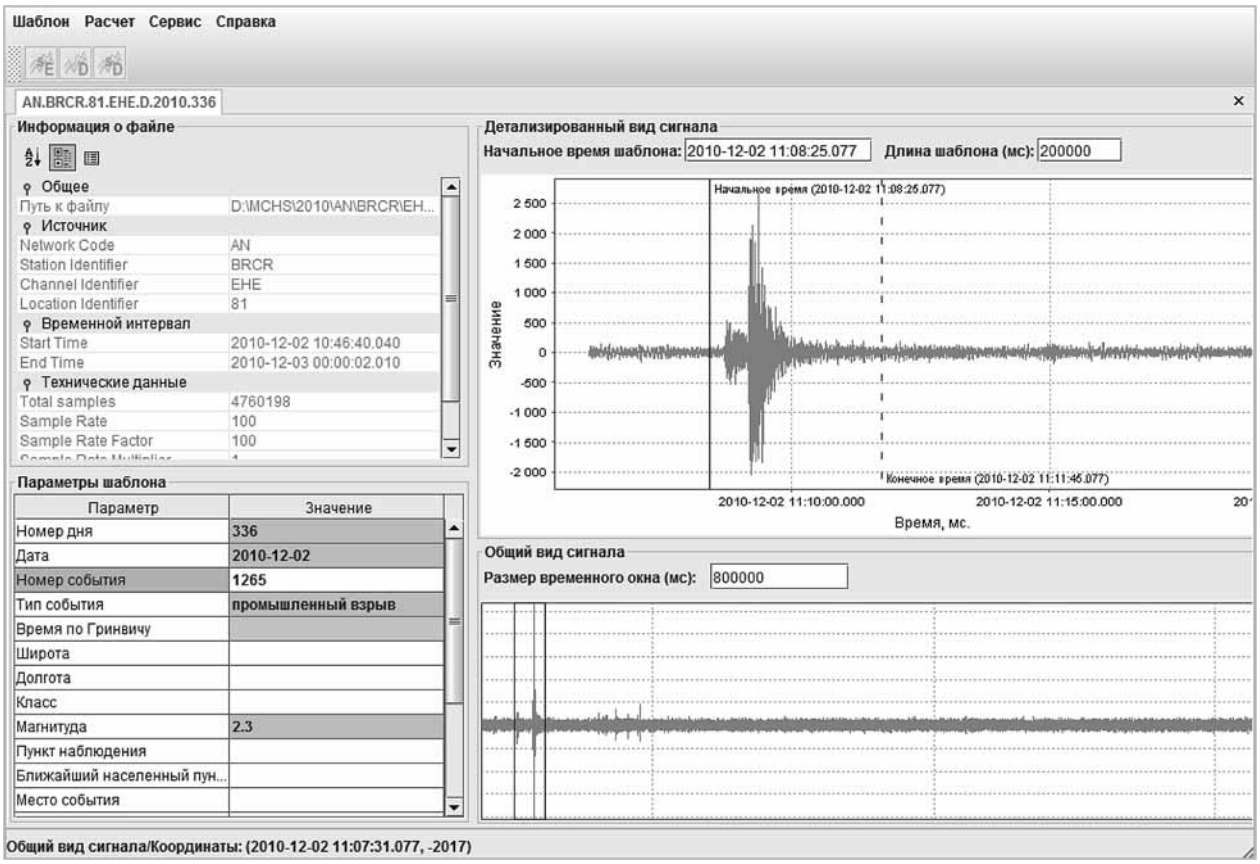

Рис. 5. Интерфейс работы с сейсмическими сигналами

Накопление данных и первичная классификация событий проводится Службой сейсмических наблюдений Агентства по защите населения и чрезвычайным ситуациям Администрации Кемеровской области. Классификация проводится по следующим типам событий: далекое землетрясение; региональное землетрясение; промышленный взрыв; горный удар; (неизвестное) сейсмическое событие. Однако следует отметить, что человеческий фактор, сильная сейсмическая зашумленность региона и плохое взаимодействие с угледобывающими предприятиями ставят под сомнение и/или оставляют без подтверждения экспертные заключения о классе событий и конкретном предприятии, если событие классифицировано как промышленный взрыв.

Программная реализация

Программный комплекс условно можно разделить на две части: стационарный модуль и веб-сервис (http://84.237.33.111:8080/seismatica.java/).

Стационарный модуль представляет собой java-приложение со следующими функциональными возможностями:

- открытие, просмотр, редактирование, выбор временного окна и установка границ обрезки сигнала справа и слева, сохранение сигналов в базе данных в виде

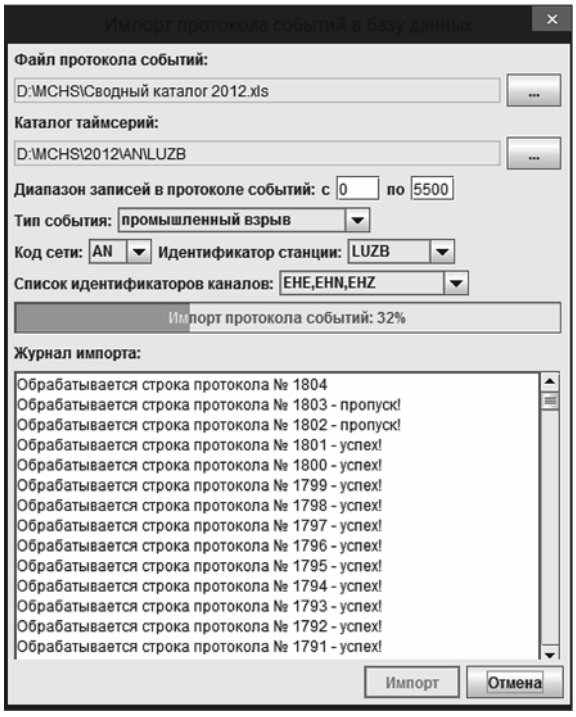

Рис. 6. Интерфейс импорта данных

шаблонов с соответствующими атрибутами (магнитуда, класс, долгота/широта, тип станции, канал и др.) (рис. 5);

- считывание в автоматическом режиме данных из протокола сейсмических наблюдений, распаковка сигнала из формата miniSEED, синхронизация каналов по глобальной временной шкале в заданном интервале каждого канала и запись шаблона в базу данных посредством DML-инструкции; распаковка файла организована посредством SEED-кодеков с применением алгоритма компрессии/декомпрессии Steim1 [9] (рис. 6).

Веб-сервис построен на базе технологий JSP и JavaScript с применением SmartClient Framework [10]. Он предоставляет следующие функциональные возможности:

- выбор шаблонов событий из базы данных посредством визуального многоуровневого SQL-конструктора (рис. 7);
- анализ моделей сигналов в графическом виде, визуализация классифицируемых событий на картах Google (рис. 8);

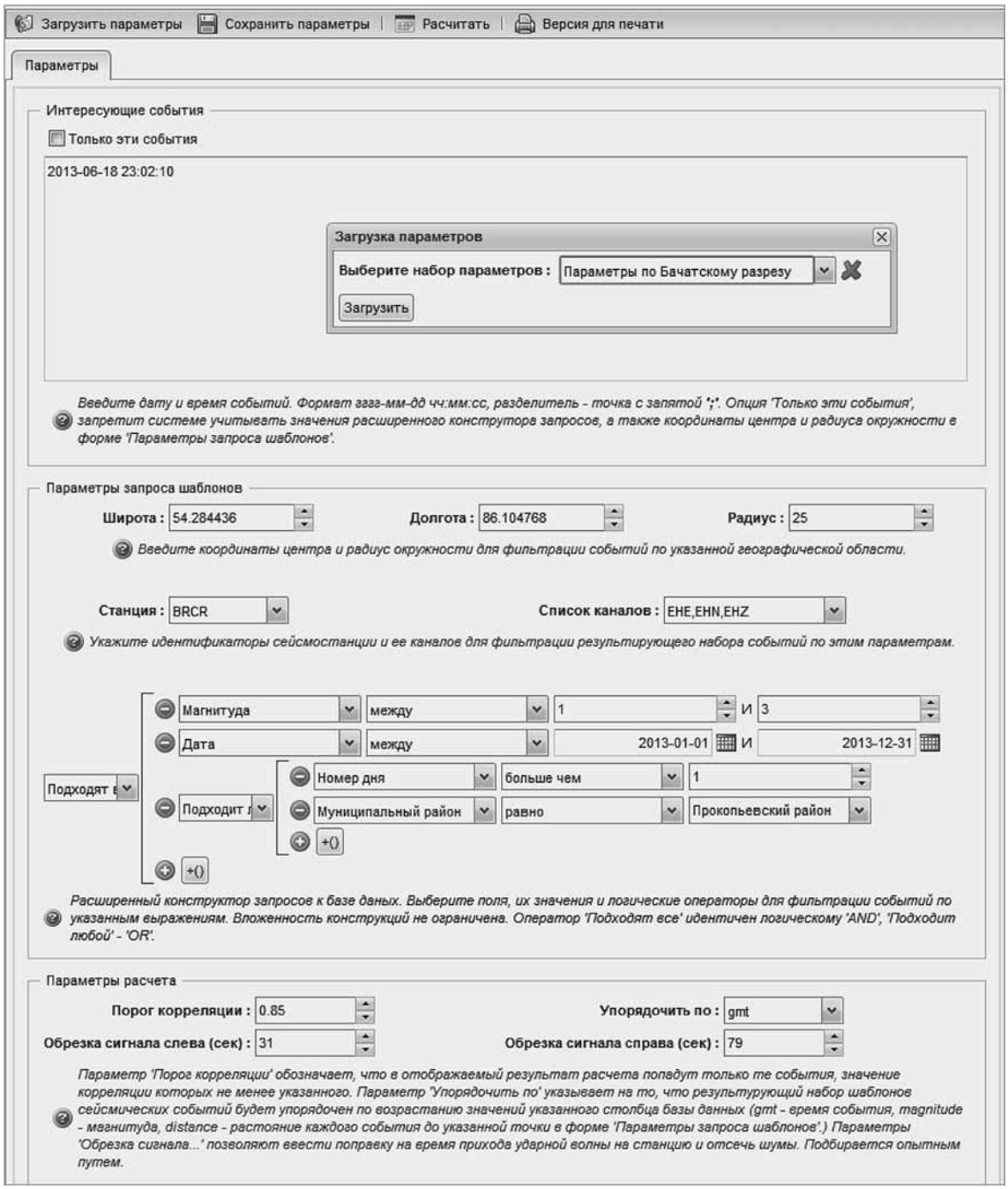

Рис. 7. Веб-интерфейс сервиса для отбора шаблонов и задания начальных параметров расчета

• загрузка/сохранение начальных параметров расчета и создание отчетов для каждого варианта расчета с возможностью печати (рис. 8);

• расчет моделей по математическим алгоритмам классификации сейсмособытий (см. выше) и фильтрация матрицы информативности по значениям коэффициентов парных корреляций.

Пример использования программного комплекса

В качестве примера рассмотрено сейсмическое событие от 21.03.2013 10:27:08.45 GMT, зарегистрированное на территории Кемеровской области и не классифицированное экспертами.

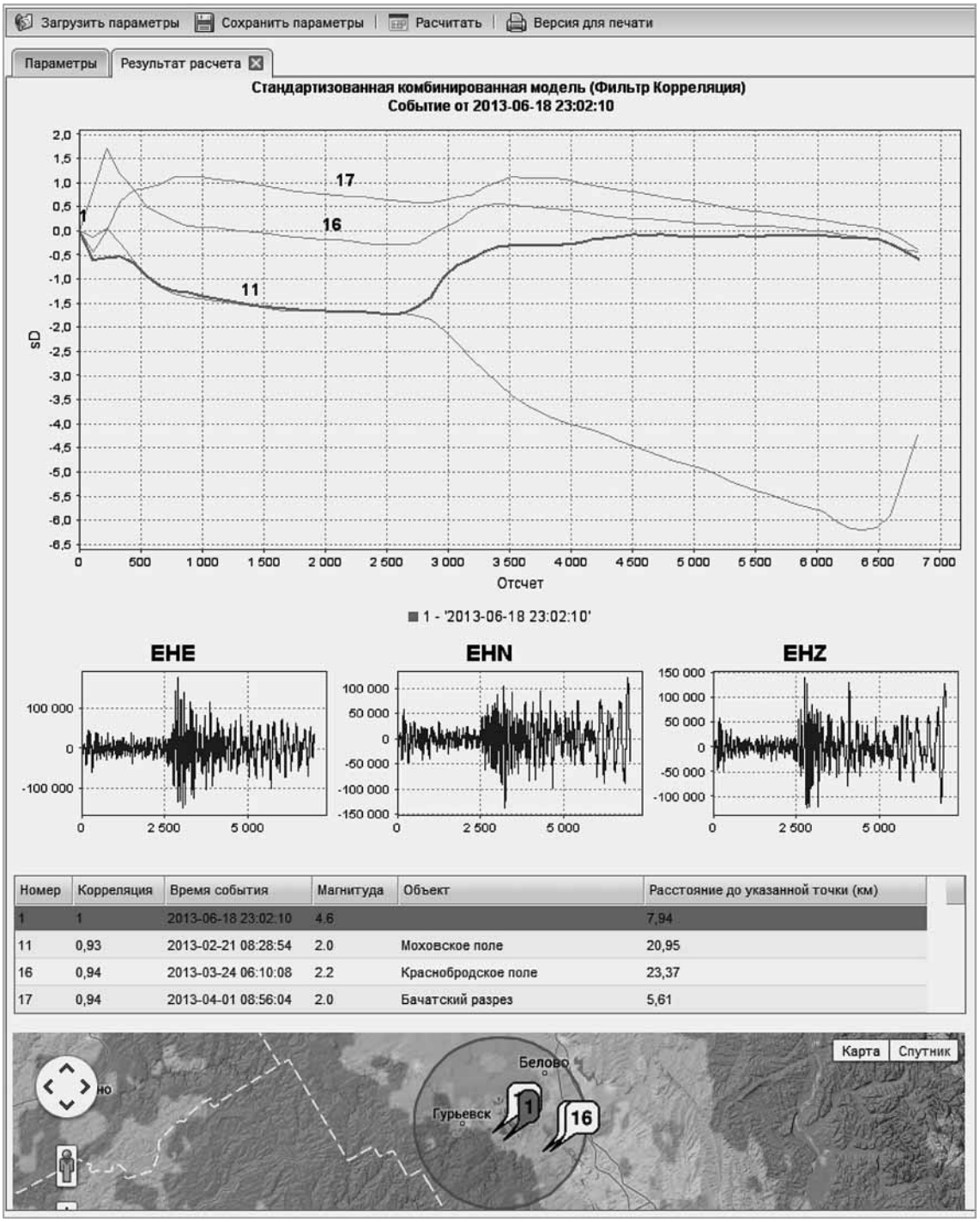

Рис. 8. Веб-интерфейс отображения результатов расчета

Через веб-интерфейс "Сейсматики" запросом получена выборка из 120 близких по магнитуде событий в радиусе 25 км от классифицируемого и подтвержденных как "промышленный взрыв", а также ряд таких же неизвестных событий. За основу взяты записи станции BRCR.

Согласно алгоритму классификации сейсмособытий рассчитана матрица информативности (рис. 9—10). После фильтрации, отображенной на рис. 10, и из таблицы корреляций (рис. 11) можно сделать заключение, что событие № 1 (классифицируемое событие) подобно трем промышленным взрывам № 27, 49, 52

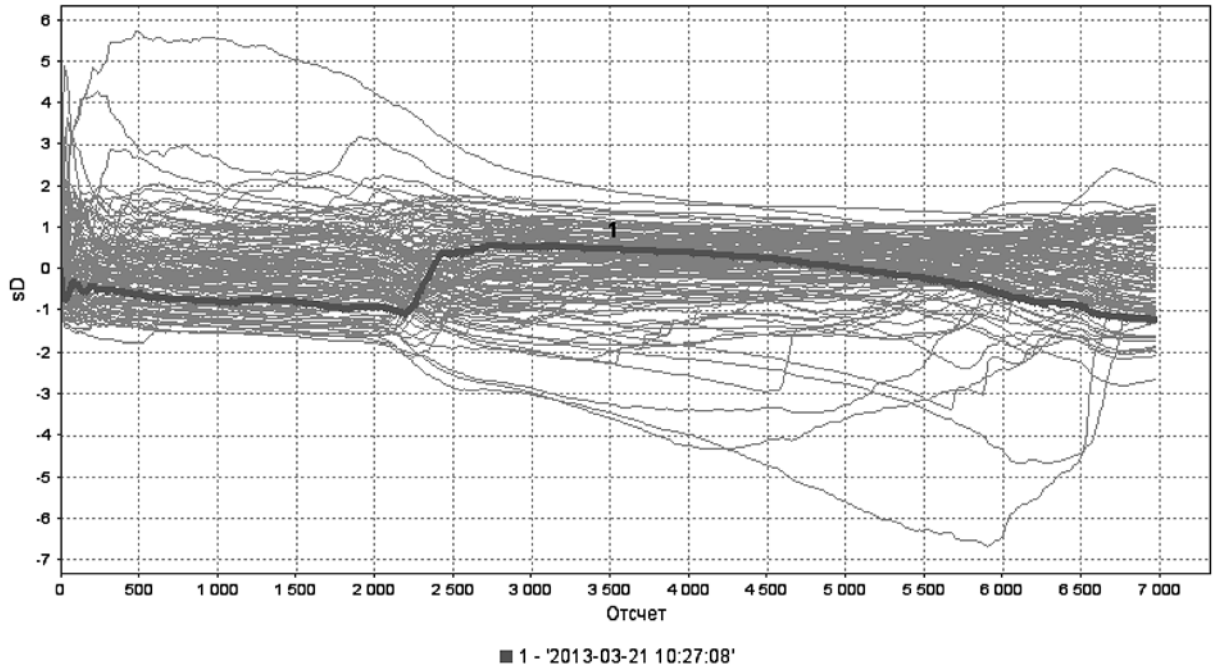

Рис. 9. Изображение матрицы информативности в виде графиков значений с выделенным классифицируемым событием:
sD — стандартизованная комбинированная модель

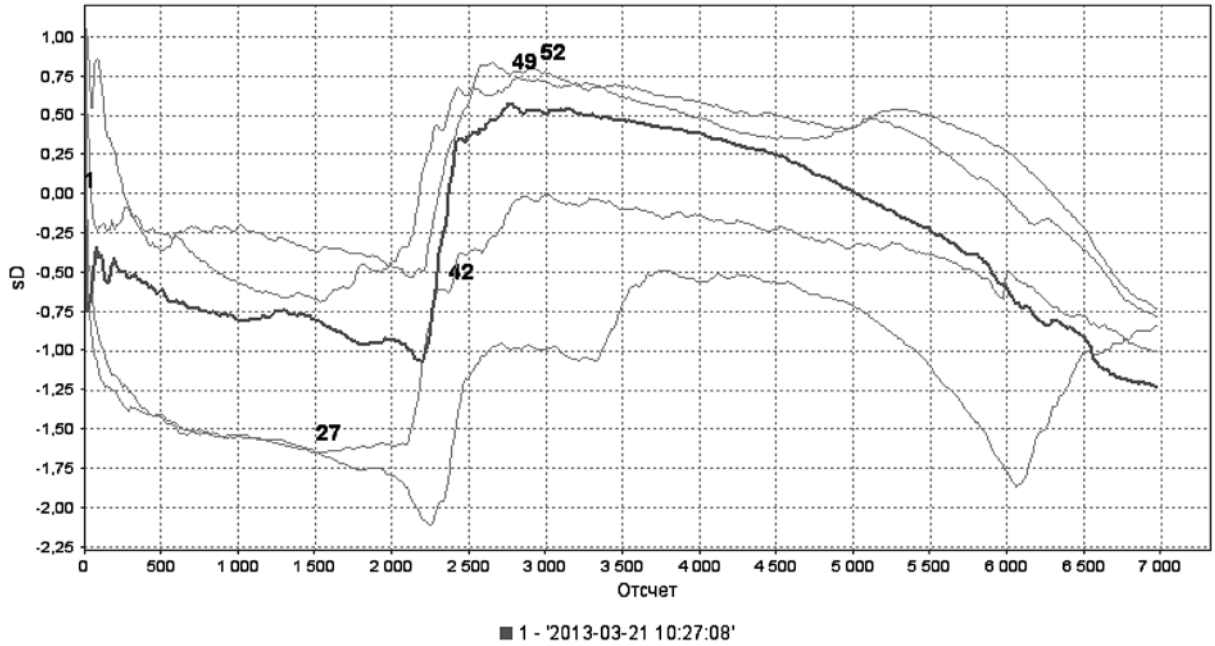

Рис. 10. Изображение матрицы информативности в виде графиков значений с фильтром по корреляции (значение корреляции ⩾ 0,9):
sD — стандартизованная комбинированная модель

Номер	Корреляция	Время события	Магнитуда	Объект	Расстояние до указанной точки (км)
1	1	2013-03-21 10:27:08	1.5		0
27	0,91	2013-02-08 07:40:31	1.9	Моховское поле	4,65
42	0,93	2013-03-06 08:09:53	2.0		15,53
49	0,9	2013-03-15 08:39:01	2.0	Моховское поле	4,64
52	0,91	2013-03-20 07:37:31	1.9	Моховское поле	6,8

Рис. 11. Таблица корреляций классифицируемого события

на Моховском поле (разрезе), а также еще одному неизвестному событию № 42.

Заключение

На основе существующих специализированных программных средств и данных наблюдений региональной сети сейсмических станций разработан функционально эффективный программный комплекс "Сейсматика" для классификации сейсмических событий на территории Кемеровской области. При этом были использованы только открытые либо GPL-среды разработки и библиотеки.

Список литературы

1. **Quality** Control of field data and initial processing of 2D and 3D seismic exploration data. [Электронный ресурс]. URL: http://www.geoleader.ru/geoseisqc/en/index.html — дата обращения 15.01.2014.

2. **Paradigm** — Stratimagic Seismic Facies Classification. [Электронный ресурс]. URL: http://www.pdgm.com/Products/Stratimagic — дата обращения 15.01.2014.

3. **Алешин И. М., Корягин В. Н., Сухорослов О. В. и др.** Инверсия сейсмических данных: высокоуровневый веб-интерфейс к инструментарию Globus Toolkit // Научно-техническая информация. Сер. 1. Организация и методика информационной работы. 2013. № 7. С. 19—22.

4. **Томилин Н. Г., Дамаскинская Е. Е., Павлов П. И.** Статистическая кинетика разрушения и прогноз сейсмических явлений // Физика твердого тела, 2005. Т. 47. Вып. 5. С. 955—959.

5. **Langer H., Falsaperla S., Powell T., Thompson G.** Automatic classification and a-posteriori analysis of seismic event identification at Soufrière Hills volcano, Montserrat // Journal of Volcanology and Geothermal Research. 2006. Vol. 153 (1). P. 1—10.

6. **Логов А. Б., Замараев Р. Ю., Логов А. А.** Анализ состояния систем уникальных объектов // Вычислительные технологии. 2005. Том 10. № 5. С. 49—53.

7. **Замараев Р. Ю.** Развитие энтропийного метода анализа процессов в системах горного производства // Горный информационно-аналитический бюллетень. 2009. Т. 7, № 12. С. 136—142.

8. **Опарин В. Н., Потапов В. П., Логов А. Б., Замараев Р. Ю., Попов С. Е.** Геоинформационная система мониторинга для регионального контроля геомеханико-геодинамической ситуации в Кузбассе на основе энтропийного подхода к анализу сейсмических записей // Физико-технические проблемы разработки полезных ископаемых. 2013. № 3. С. 148—156.

9. **USC Seismology.** [Электронный ресурс]. URL: http://www.seis.sc.edu/ — дата обращения 21.01.2014.

10. **SmartClient** Ajax RIA System ‖ Isomorphic Software. [Электронный ресурс]. URL: http://www.smartclient.com — дата обращения 21.01.2014.

ИНФОРМАЦИЯ

УДК 004.912

В. А. Васенин, д-р физ.-мат. наук, проф., зав. отделом, Институт проблем информационной безопасности МГУ, e-mail: vasenin@msu.ru,
С. А. Афонин, канд. физ.-мат. наук, вед. науч. сотр., НИИ механики МГУ имени М. В. Ломоносова, e-mail: serg@msu.ru,
Д. С. Панюшкин, аспирант МГТУ им. Н. Э. Баумана, вед. инженер-программист, ЗАО "Expert Solutions", email: dspan@yandex.ru

Модели распространения информации в социальных сетях: тестовые испытания

Настоящая статья является продолжением ранее опубликованной в журнале "Программная инженерия" статьи "Модели распространения информации в социальных сетях" [1]. Представлены результаты тестирования алгоритма распространения информации в социальных сетях и web-блогах, который описан в работе [1]. Тестирование проведено как на синтетических, так и на реальных данных социальной сети Twitter.

Ключевые слова: *социальные сети, пути распространения информации, модели информационных потоков, источники информации, Twitter*

V. A. Vasenin, S. A. Afonin, D. S. Panushkin

Models of Information Dissemination in Social Networks: Experoimental Results

This article contains experimental results for the information propagation model described in "Models of Information Dissemination in Social Networks" article published in this journal [1]. Experiments were conducted on both synthetic and real-world data collections.

Keywords: *social network, information propagation, information stream model, information sources, Twitter*

Введение

В первой части данной работы была описана модификация алгоритма [2] выявления путей распространения информации в социальных сетях, которая использует более точную математическую модель. В данной работе приводятся результаты тестирования предложенного алгоритма, включающие тестирование как на синтетических, так и на реальных данных.

Следует отметить, что тестирование алгоритмов, работающих с информацией социальных сетей, связано со значительными сложностями, так как в большинстве случаев правильный результат работы неизвестен и может носить субъективный характер. Например, для рассматриваемой задачи распространения информации невозможно установить, откуда именно тот или иной пользователь реально получил сведения о наступившем событии. Дополнительную неопределенность вносит и задача сопоставления двух текстовых сообщений в целях определения их тематической направленности. Очевидно, что далеко не всегда возможно однозначно определить, что два сообщения относятся к одной теме.

В рамках работы, описанной в статье, тестирование алгоритма проводилось в двух вариантах. Первый вариант состоит в тестировании на синтетических данных. В этом случае "загадывается" правильная схема распространения сообщений, которая подается на вход программной реализации алгоритма (возможно, с добавлением некоторого шума). Вторая схема тестирования использует данные из социальной сети Twitter. В силу технических особенностей данной сети (сообщения, как правило, копируют одним нажатием кнопки) можно ожидать, что скопированные сообщения не изменяются и содержат ссылку на первоисточник (идентификатор первого сообщения). Например, если пользователь *C* копирует сообщение у пользова-

теля *B*, которое ранее было скопировано у пользователя *A*, то в сообщении *C* будет стоять ссылка на *A*, но информация о пользователе *B* (и любых других пользователях между *A* и *C*) будет недоступна. Вместе с тем существует возможность получить "окружение" пользователя — список тех пользователей, сообщения которых поступают в его список входящих сообщений. В следствие технических особенностей данной социальной сети окружение пользователя является приближением множества возможных источников информации пользователя. Все перечисленное делает сеть Twitter интересным объектом для тестирования алгоритмов распространения информации.

Программная реализация алгоритма (далее по тексту — алгоритма), на которой проводились тестовые испытания, написана на языке C++. Данные о сообщениях сети Twitter записываются в базу данных SQLite.

В статье представлены сравнительные результаты тестирования алгоритма, приведенного в работе [2], и алгоритма, предлагаемого в данной работе. Приведены результаты тестирования нового алгоритма на синтетических и реальных данных. Для описания условий тестирования и комментирования их результатов использованы обозначения, соответствующие принятым в работах [1, 2].

1. Тестирование на синтетических данных

В работе [2] описано тестирование, которое проводилось на случайных графах с фиксированной исходящей степенью вершины, определяемой как число исходящих ребер. Каждому ребру назначались одинаковые вероятности $r_{u,v} = 2/3$ и $k_{u,v} = 1/10$. Затем эмулировалось распространение сообщений по графу. Каждая вершина генерировала от 20 до 60 уникальных тем. Далее, за каждый новый день вершины, смежные с "зараженными" новой темой вершинами, читали сообщение с вероятностью $r_{u,v}$ и при прочтении копировали с вероятностью $k_{u,v}$. Если вершина прочитала сообщение заданной темы, она помечалась как уже рассмотренная и далее не могла прочитать сообщения этой темы. Процесс распространения завершался, когда все вершины, смежные с "зараженными", уже прочитали тему, но ни одна из них не скопировала ее за текущий день.

На рис. 1—3 приведены графики распределения числа тем сообщений по числу вершин, которые посетила тема, для нескольких из графов, на которых проводилось тестирование.

Очевидно, что в заданных условиях весь граф определяется как один кластер. В модели не отражены также наличие новостных сообщений и схожесть поведения пользователей. Поэтому оценивать целесообразно лишь точность восстановления ребер исходного графа. Так как базовый алгоритм работает с полным графом, сравнение нового алгоритма проводится с модификацией базового, в которой ребро от u к v удаляется, если $\Sigma_{m \in Sl} p_{u,v}(m) < 10^2$. Значение параметра β,

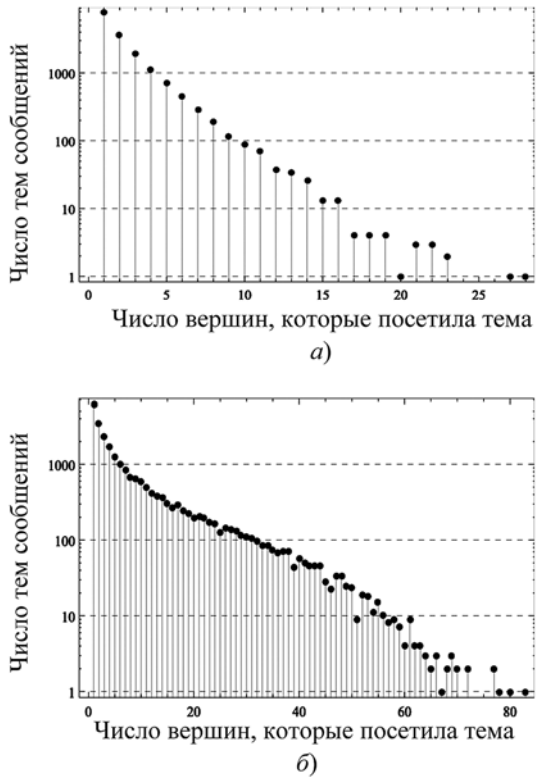

Рис. 1. Распределение числа тем сообщений по числу вершин, которые посетила тема: 1000 вершин графа; степень вершины 5 (*а*) и 9 (*б*)

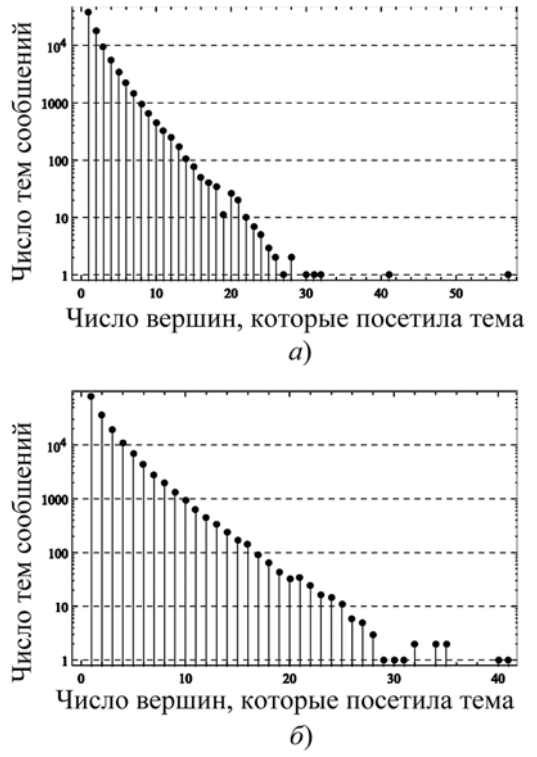

Рис. 2. Распределение числа тем сообщений по числу вершин, которые посетила тема: степень вершины 5; число вершин графа 5000 (*а*) и 10 000 (*б*)

Таблица 1

Результаты тестирования

N_v	d	N_m	t	E	r	k	E_{err}	r_{err}	k_{err}	№
100	3	4022	18	297	0,47	0,1	36	0,9	0,008	1
100	3	4022	18	287	0,46	0,1	8	0,86	0,01	2
100	5	4002	18	498	0,5	0,1	98	0,91	0,004	1
100	5	4002	18	484	0,5	0,1	26	0,9	0,05	2
100	9	4170	20	900	0,57	0,08	399	0,8	0,03	1
100	9	4170	20	885	0,57	0,09	206	0,85	0,03	2
500	3	19 199	19	1481	0,48	0,1	185	0,92	0,01	1
500	3	19 199	19	1444	0,47	0,1	23	0,8	0,01	2
500	5	19 621	20	2492	0,52	0,1	465	0,87	0,004	1
500	5	19 621	20	2407	0,51	0,1	59	0,88	0,003	2
500	9	19 746	30	4499	0,6	0,09	1393	0,63	0,002	1
500	9	19 746	30	4347	0,6	0,1	343	0,72	0,002	2
1000	3	40 059	20	2966	0,47	0,1	362	0,88	0,007	1
1000	3	40 059	20	2906	0,47	0,1	42	0,87	0,006	2
1000	5	40 167	21	4979	0,52	0,1	739	0,88	0,005	1
1000	5	40 167	21	4855	0,51	0,11	88	0,86	0,005	2
1000	9	39 910	28	9000	0,61	0,1	2344	0,6	0,002	1
1000	9	39 910	28	8750	0,6	0,1	417	0,66	0,002	2
5000	3	200 336	21	14 854	0,47	0,11	1600	0,9	0,007	1
5000	3	200 336	21	14 487	0,47	0,11	159	0,83	0,006	2
5000	5	199 774	30	24 918	0,52	0,11	4015	0,88	0,004	1
5000	5	199 774	30	24 255	0,51	0,11	430	0,83	0,005	2
5000	9	201 139	45	44 999	0,62	0,1	12507	0,56	0,01	1
5000	9	201 139	45	43 751	0,62	0,1	835	0,58	0,01	2
10 000	3	399 860	22	29 690	0,48	0,11	3188	0,9	0,007	1
10 000	3	399 860	22	29 065	0,47	0,11	337	0,81	0,007	2
10 000	5	401 371	25	49 839	0,52	0,11	7918	0,87	0,004	1
10 000	5	401 371	25	48 479	0,51	0,11	797	0,8	0,005	2
10 000	9	397 921	49	—	—	—	—	—	—	1
10 000	9	397 921	49	—	—	—	—	—	—	2

Рис. 3. Распределение числа тем сообщений по числу вершин, которые посетила тема: 5000 вершин графа, степень вершины 9

определяющего критерий остановки процесса построения минимальных покрытий, полагается равным 0,7.

В табл. 1 представлены сравнительные результаты тестирования. При описании результатов использованы следующие обозначения: N_v — число вершин; d — исходящая степень вершины; N_m — число тем сообщений; t — продолжительность процесса распространения в днях; E — число верно восстановленных ребер; E_{err} — число неверно восстановленных ребер; r — средняя вероятность чтения по верно восстановленным ребрам; k — средняя вероятность копирования по верно восстановленным ребрам; r_{err} — средняя вероятность чтения по неверно восстановленным ребрам; k_{err} — средняя вероятность копирования по неверно восстановленным ребрам; № — номер алгорит-

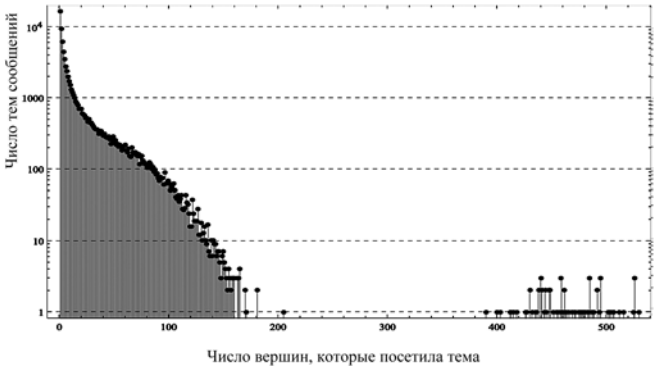

Рис. 4. Распределение тестовых данных: в левой части оказались сгруппированы данные по обычным сообщениям, а в правой — по новостным

ма (1 в случае старого, 2 в случае нового). Прочерки означают превышение лимита оперативной памяти в 8 Гбайт при выполнении алгоритма.

Тестирование показало, что базовый алгоритм восстанавливает 99 % ребер верно, а 10...20 % от этого числа — неверно. Новый же алгоритм восстанавливает 96 % ребер верно, а 1...3 % от этого числа — неверно. Точность восстановления $r_{u,v}$ и $k_{u,v}$ для двух алгоритмов одинакова: значение $k_{u,v}$ восстанавливается с большой степенью точности, однако для $r_{u,v}$ можно говорить лишь о точном определении порядка значения.

Тестирование нового алгоритма проводилось на случайном графе, имеющем 3000 вершин. Он разбит на два кластера в 1000 и 2000 вершин. Каждая вершина имеет девять исходящих ребер внутри своего кластера, а одно ребро ведет из нее в другой кластер. Каждому ребру присваивается вероятность чтения $r_{u,v} = 2/3$ и вероятность копирования $k_{u,v} = 1/10$. Каждой вершине присваивается вероятность k_v, которая выбирается случайным образом из множества {0,1, 0,2, 0,3}. Каждая вершина генерирует от 20 до 60 уникальных тем, которые распространяются по алгоритму, взятому из предыдущего тестирования. Далее случайным образом выбирают 100 вершин, из которых распространяется по одной *новостной* теме. Распространение новостной темы аналогично распространению обычной, за исключением того, что в качестве вероятности $k_{u,v}$ выступает k_v.

На рис. 4 представлен график распределения числа тем сообщений по числу вершин, которые посетила тема.

При работе алгоритм разбил граф на два кластера в 1058 и 1942 вершины. В табл. 2 приведены численные результаты теста (использованы обозначения из табл. 1).

Как можно заметить, средняя вероятность копирования новостных сообщений составляет 0,16, что приближенно оценивает математическое ожидание значения k_v (равное 0,2), которое использовалось при генерации данных.

2. Тестирование на реальных данных

Как уже отмечалось ранее, в качестве источника реальных данных выступает социальная сеть Twitter. В данном разделе описывается процедура сбора первичных данных и приводятся результаты тестирования предлагаемого алгоритма. Сравнение с исходным алгоритмом [2] не проводилось, так как в случае обработки реальных данных возможность объективного сравнения результатов затруднена.

2.1. Задача сбора данных

Число пользователей типичной социальной сети измеряется в миллионах, и каждый из этих пользователей ежедневно генерирует сообщения. Владельцы же ресурсов социальных сетей всячески ограничивают доступ к получению больших объемов данных с их серверов. Отсюда возникает задача эффективной организации сбора данных для тестирования.

Постановка задачи: требуется собрать статистический материал в виде текстовых сообщений пользователей, минимизировав при этом сетевой трафик и число запросов к серверу владельцев социальной сети, при условии, что в процессе опроса каждого пользователя сервер выдает не более чем N последних его сообщений.

Следует отметить, что регулярный опрос всех пользователей с некоторой фиксированной частотой не удовлетворяет требованию минимизации трафика. Для решения этой задачи вводится следующая математическая модель.

- Предполагается, что пользователи создают сообщения независимо друг от друга.
- Процесс генерации пользователем сообщений представляется как поток однородных событий без последействия.
- На участке времени, близком к моменту последнего опроса пользователя, процесс генерации сообщений представляется в виде стандартного Пуассоновского потока, определяемого своей *интенсивностью* λ. Физический смысл параметра λ — среднее число событий за единицу времени.

За λ^u можно принять среднее число сообщений в день, "производимых" пользователем u. Для его оценки подсчитывается набор $\lambda_1^u, \ldots, \lambda_n^u$ методом "плавающего окна": весь отрезок времени наблюдения за

Таблица 2

Результаты синтетического теста

N_v	d	N_m	t	E	r	k	E_{err}	r_{err}	k_{err}	k_v
3000	10	120 059	38	29 243	0,64	0,1	1600	0,54	0,0006	0,16

пользователем разбивается на n перекрывающихся интервалов, и на каждом из них подсчитывается среднее число сообщений за день — λ_i^u. Далее λ^u приравнивается к средневзвешенной сумме, где $\omega_1, ..., \omega_n$ — набор возрастающих весов. Таким образом, чем ближе интервал построения λ_i^u к моменту последнего опроса пользователя, тем с большим весом λ_i^u войдет в сумму.

Далее можно рассмотреть $M < N$ — число сообщений, которое планируется получить за один опрос пользователя. Для каждого пользователя определяется свой момент времени следующего опроса, который отстоит от момента предыдущего опроса t_{last}^u на $\Delta t^u = \frac{M}{\lambda^u}$.

Из теории вероятностей известно, что вероятность получения на промежутке от предыдущего до следующего опроса числа сообщений больше чем N равна

$$Pr[k > N] = \sum_{k=N+1}^{\infty} \frac{M^k}{k!} \exp\{-M\},$$

где $Pr[k > N]$ — вероятность того, что $k > N$.

Таким образом, алгоритм сбора данных для тестирования заключается в вычислении для каждого опрашиваемого пользователя времени следующего опроса $t^u = t_{last}^u + \Delta t^u$ по предложенной выше схеме, и опроса пользователей в очередности возрастания t^u. Оптимальным образом выбрав M в зависимости от N можно получить достаточно малую вероятность потери сообщений при таком алгоритме опроса.

2.2 Описание исходных данных

Исходные данные для тестирования алгоритма были взяты с web-ресурса twitter.com. REST API twitter.com [3] позволяет за один запрос к серверу получить не более 100 последних ретвитов[1] для пользователя, определенного своим идентификационным номером. Также API позволяет получить идентификационные номера друзей данного пользователя.

В процессе сбора данных для каждого обработанного пользователя сохранялись его последние ретвиты и множество его друзей. Далее в очередь на обработку добавлялись те пользователи, чьи сообщения наблюдались у текущего пользователя в качестве ретвитов. На обрабатываемых пользователей накладывались следующие ограничения: пользователь должен быть русскоязычным, иметь не менее 200 читателей и сам читать от 30 до 1000 других пользователей. Стартовав с одного пользователя, процесс за две недели обработал около 30 тыс. пользователей, удовлетворяющих поставленным критериям. После этого скорость добавления в очередь опроса новых пользователей резко упала, что стало поводом к остановке процесса сбора данных.

Треть пользователей из означенных 30 тыс. пришлось отбросить, так как число их ретвитов за весь период существования не превышало 10, что недостаточно для построения оценок. Для оставшихся 20 тыс. пользователей было собрано около 1,5 млн ретвитов, относящихся примерно к 1,2 млн различных тем. Самое раннее сообщение было создано 6 июля 2006 г., самое позднее — 4 мая 2012 г. Средняя дата создания сообщения — 10 января 2012 г. К сожалению, статистика могла быть несколько искажена тем, что не имеющие ретвитов сообщения не собирались. Из собранных 1,2 млн тем около 900 тыс. имели в качестве первоисточника пользователя, не относящегося к множеству опрошенных, что обусловило необходимость введения отдельной вершины графа, отождествляемой с "внешним миром". Эта вершина была объявлена первоисточником данных 800 тыс. сообщений.

На рис. 5 приведен график распределения числа тем сообщений по числу вершин, которые посетила тема. Таким образом, приблизительно 10^6 сообщений появились только в одной вершине, и несколько десятков сообщений появились в 50 и более вершинах.

Как и в синтетическом тесте с новостными сообщениями, на графике наблюдается высокий уровень сгруппированности данных по обычным сообщениям в левой части и некоторые уплотнения в правой части, отделенные от левой пустым промежутком. Хотя эффект не столь выражен как при тестировании на синтетических данных, эти уплотнения соответствуют именно новостным сообщениям. Например, наибольший охват (более 300 вершин) получили следующие сообщения:

- "Siteko — надежный #хостинг от 35 руб. и #VDS от 84 руб. за месяц. Бесплатный перенос сайта. Тестовый период 10 дней. http://t.co/WOn6xOlY"
- "Персональный гороскоп на 2012 год http://t.co/o42p0sym"

Как можно заметить, они относятся к вирусной рекламе или объявлениям, а значит, к *новостным* сообщениям.

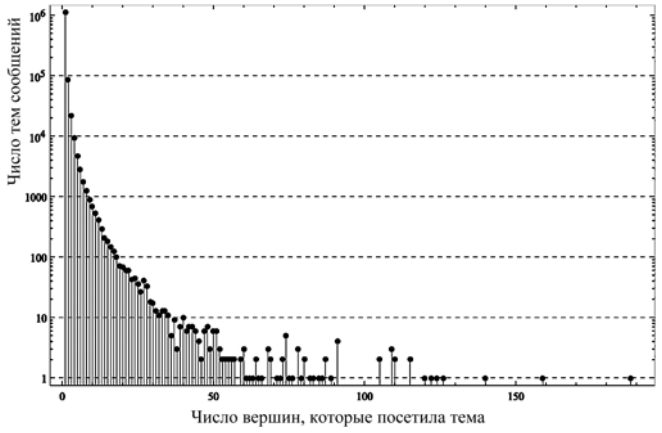

Рис. 5. Распределение числа тем сообщений по числу вершин для реальных данных

[1] Ретвитом (от английского *retweet*) называют сообщение, которое получено копированием, обычно дословным, сообщения другого пользователя. Данный механизм является основой для передачи информации в этой социальной сети, так как технически может быть выполнен одним нажатием кнопки.

2.3. Результаты тестирования

В процессе тестирования было обработано 20 594 вершины и 1 523 779 сообщений, относящихся к 1 253 441 теме. Из этих сообщений 1 114 153 имеют источником "внешний мир" и относятся к 913 818 темам. Все вершины оказались выделены в один кластер. Этот факт согласуется с критериями построения множества опрашиваемых пользователей. Восстановлено 273 184 ребра, по которым могли проходить сообщения. Из этих ребер 122 914 совпадают с формальными отношениями дружбы между пользователями, а 114 270 не совпадают. Обнаружение столь большого числа ребер, не присутствующих в графе друзей, можно объяснить наличием во внешнем мире вершин-посредников, через которые передается информация между некоторыми вершинами кластера.

Среднее значение $r_{u,\,v}$ составило 0,94 с дисперсией 0,03. Этот факт означает, что большинство пользователей читает Twitter ежедневно. Среднее значение $k_{u,\,v}$ составило 0,017 с дисперсией 0,0026, что означает низкую вероятность ретвита. Среднее значение толерантности к новостным сообщениям k_v составило 0,074 с дисперсией 0,019. Это обстоятельство демонстрирует предпочтение пользователей к ретвиту новостных сообщений в сравнении с ретвитом обычных.

На рис. 6—8 приведены графики, описывающие распределения различных параметров построенного графа распространения сообщений.

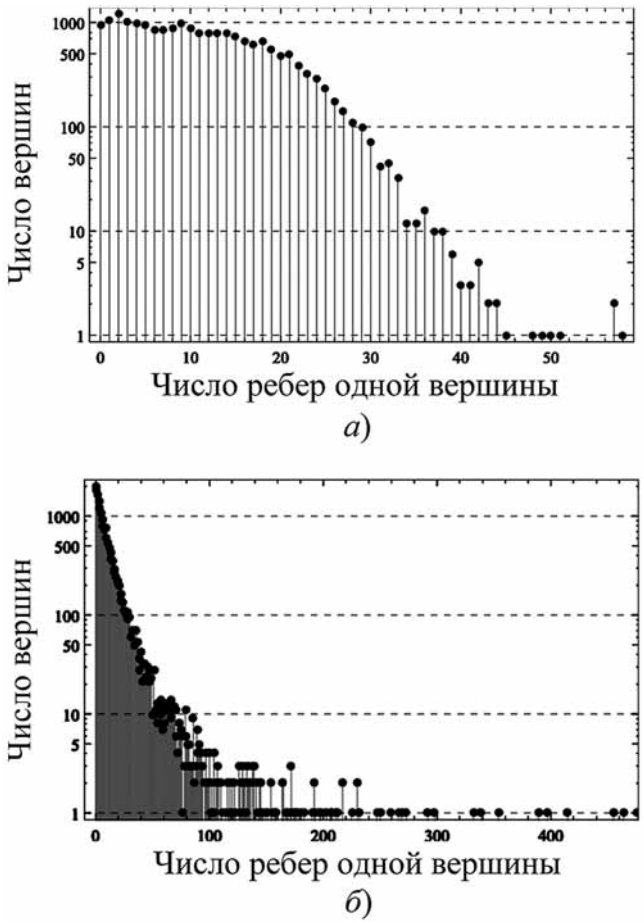

Рис. 6. Распределение числа входящих (*а*) и исходящих (*б*) ребер

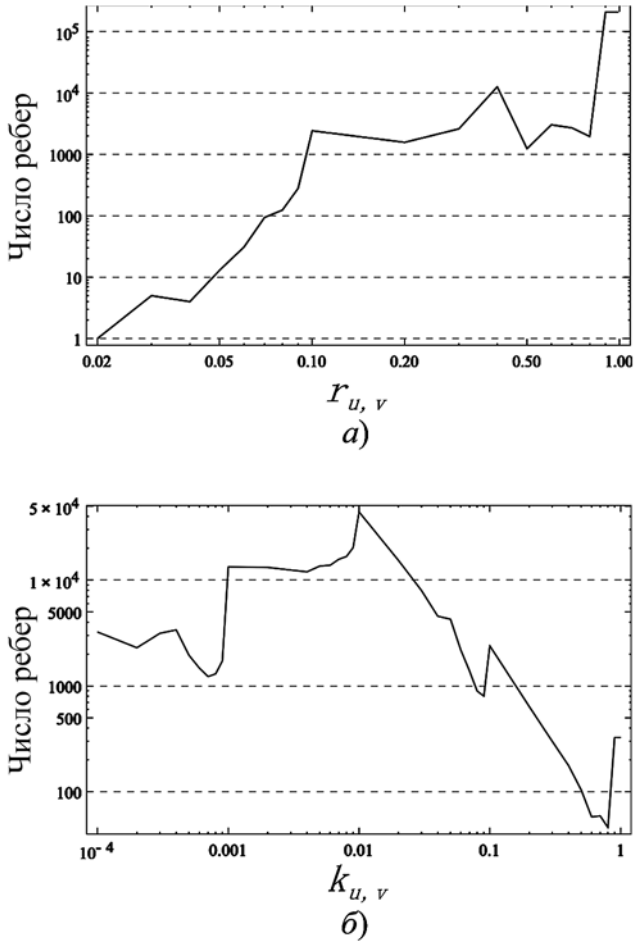

Рис. 7. Распределение значений $r_{u,\,v}$ (*а*) и $k_{u,v}$ (*б*)

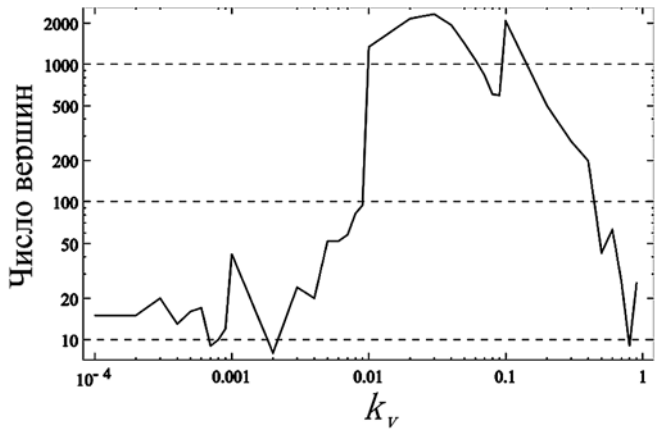

Рис. 8. Распределение значений k_v

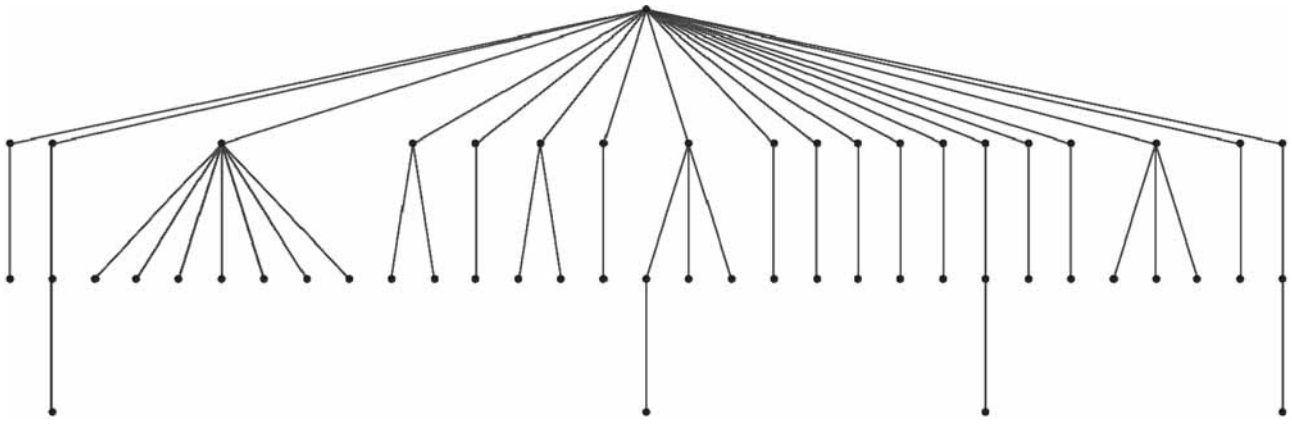

Рис. 9. Дерево распространения сообщения

Интересными представляются также связи пользователей с вершиной W, обозначающей "внешний мир". Среднее значение $r_{W,\,v}$ составило 0,9999 с дисперсией 10^{-7}. Среднее значение $k_{u,\,v}$ составило $6{,}45 \cdot 10^{-5}$ с дисперсией $4{,}8 \cdot 10^{-9}$. Таким образом, все пользователи регулярно читают "внешний мир", а столь низкое значение вероятности копирования объясняется объединением всех внешних источников в один, который имеет очень большое число сообщений. Каждый пользователь регулярно читает лишь несколько источников, относящихся к внешнему миру, но модель предполагает прочтение всех сообщений из вершины W. Чтобы узнать истинное значение $k_{W,\,v}$, его оценку следует умножить на число источников, составляющих "внешний мир", и разделить на среднее число друзей пользователей.

В заключение можно привести пример построения вероятных путей распространения трех наиболее популярных сообщений (рис. 9).

В предложенной схеме описывается путь распространения сообщения, получившего максимальный охват пользователей. Корнем дерева является вершина "внешний мир". От него отходит еще около 250 потомков, из которых сообщение больше никуда не распространилось. Чтобы сохранить место, данные вершины на рис. 9 не показаны.

Заключение

Предложена формализация задачи определения характерных путей распространения информации в социальных сетях и web-блогах. Построена математическая модель, отражающая динамику распространения сообщений в социальных сетях. Предложены алгоритмы, позволяющие улучшить точность или производительность существующих методов решения задачи. Проведенное сравнительное тестирование предложенного метода решения и исходного алгоритма на синтетических данных показало, что модифицированный алгоритм обладает большей точностью (1...3 % ошибок против 10...20 % у исходного алгоритма) с сохранением полноты восстановленных путей передачи информации (96 % против 99 %).

Список литературы

1. **Васенин В. А., Афонин С. А., Панюшкин Д. С.** Модели распространения информации в социальных сетях // Программная инженерия. 2014. № 2. С. 33—42.

2. **Gruhl D., Guha R., Liben-Nowell D., Tomkins A.** Information diffusion through blogspace // *Proceedings of the 13th international conference on World Wide Web*. 2004. P. 491—501.

3. REST API v1.1 Resources.URL: https://dev.twitter.com/docs/api/1.1

ИНФОРМАЦИЯ

13—16 октября 2014 г. в Объединенном институте ядерных исследований (г. Дубна) при поддержке Российского фонда фундаментальных исследований, Института проблем информатики РАН, Московской секции ACM SIGMOD будет проводиться очередная XVI Всероссийская научная конференция с международным участием RCDL-2014 — «Электронные библиотеки: перспективные методы и технологии, электронные коллекции».

В рамках конференции RCDL-2014 планируется проведение традиционного семинара молодых ученых "Диссертационные исследования по тематике информационных технологий, связанных с электронными библиотеками", на котором авторам работ, отобранных на основе предварительного рецензирования, будет дана возможность представить текущие результаты своих исследований, а также обсудить их сильные и слабые стороны с более опытными коллегами.

Подробную информацию о конференции можно найти на сайте ***http://rcdl2014.jinr.ru***

УДК 004.5

С. Ф. Сергеев, д-р психол. наук, проф., Санкт-Петербургский государственный университет, вед. науч. сотр., ГНЦ ЦНИИ робототехники и кибернетики, г. Санкт-Петербург, e-mail: ssfpost@mail.ru

Методологические вопросы пользовательского интерфейса информационных систем

Статья посвящена исследованию методологических вопросов разработки пользовательского интерфейса информационных систем. Рассмотрены методологические, теоретические, психологические и технологические задачи, особенности их решения в зависимости от уровня организации и степени интеллектуализации процессов взаимодействий в информационных системах.

Ключевые слова: *интерфейс, пользовательский интерфейс, информационная система, техногенная среда, гибридный интеллект, искусственный интеллект, диффузный интеллект, интеллектуальные среды и системы*

S. F. Sergeev

Methodological Problems of the User Interface Information Systems

In current article we present our view of the human interfaces development problem. We analyze methodological, theoretical, psychological and technological difficulties and the ways of solving these difficulties, depending on the intellectual degree of interaction processes in information systems.

Keywords*: interface, user interface, information system, technogenic environment, hybrid intelligence, artificial intelligence, diffuse intelligence, intelligent environments and systems*

Введение

Понятие "интерфейс" отражает формы, средства и возможности обеспечения взаимодействия двух или более систем (их компонентов) между собой независимо от их физической или ментальной природы в процессе достижения их целей. В настоящее время это понятие является чрезвычайно популярным и широко распространенным при обсуждении в научной среде термином, имеющим междисциплинарную природу. В последнее десятилетие наблюдается его экспансия в область информационных технологий, инженерной психологии и эргономики, где наибольшее распространение получили термины "пользовательский интерфейс" и "человеко-машинный интерфейс" [1]. Однако всего лишь тридцать лет назад проблемы интерфейса в инженерной психологии практически не существовало, а основные элементы данного единства рассматривались инженерами-проектировщиками как независимые элементы информационной модели человеко-машинной системы, выполняющие функции ввода-вывода информации и формирования управляющих воздействий [2].

Можно выделить ряд этапов эволюции данного понятия в классической, неклассической и постнеклассической инженерной психологии, связанных с развитием техногенной формы человеческой цивилизации. На первом этапе проблема взаимодействия человека с орудием труда решалась на уровне обеспечения физического взаимодействия тела человека с техникой (органами управления) и входила в сферу интересов эргономики, которая и по настоящее время ассоциируется с организацией удобного рабочего места. Под интерфейсом понималась механическая, линейная моторно-перцептивная связь человеческого тела с органами управления и отображения информации.

Эстафету эргономического проектирования продолжила классическая инженерная психология, которая сосредоточила внимание на вопросах проектирования эффективного информационного взаимодействия человека-оператора с эргатической человеко-машинной системой [3]. Развитие инженерно-психологических представлений о человеко-машинном взаимодействии привело к появлению понятий "пользователь" и "пользовательский интерфейс" [4].

На неклассическом этапе объектом изучения стали интерфейсы сложных эргатических систем и техногенных сред, включающие коммуникативные самоорганизующиеся процессы и элементы технологий искусственного интеллекта [5]. Под интерфейсом стали понимать технологию общения человека с компьютером или другими системами. Объектом изучения в проблеме интерфейса стали организация и развитие кооперативных эффектов в искусственных средах, включающих глобальные информационные сети [6].

Постнеклассический этап развития взглядов на интерфейс связан с использованием системно-философских и естественно-научных представлений радикального конструктивизма, кибернетики второго порядка и исторических развивающихся систем [7].

Классическая методология проектирования интерфейса

В классическом инженерно-психологическом представлении управление эргатическими системами осуществляется посредством органов управления и пультовой аппаратуры, содержащей средства индикации и контроля, отображения информации о текущем состоянии управляемой системы и ее элементов. Возникающая информационная модель погружает оператора во взаимодействие с системой управления в рамках выполнения фиксированных алгоритмических действий с заранее ясными целями и наблюдаемыми следствиями в поведении управляемой системы [8]. Системы интерфейса, связывающие оператора с контуром управления, решают задачу преобразования управляющих воздействий в команды, результат которых выражается в наблюдаемых в системах индикации и отображения информации изменениях параметров управляемой системы. Основная задача согласования возникающих в системе "человек—машина" взаимоотношений лежит в области формирования у оператора соответствующей назначению эргатической системы концептуальной модели и навыков управления с помощью органов управления [9] и относится к инженерной психологии. Решается задача согласования психофизиологических возможностей человека с техникой [10].

Традиционно решение данных задач лежит в области психологии обучения и достигается методами профессиональной подготовки на тренажерах, содержащих модели реальной деятельности [11]. Многократное повторение профессиональной задачи приводит к появлению обучающей среды, ведущей к формированию наиболее эффективного для данного оператора способа управления [12]. По окончании обучения задача обеспечения эффективной связи "человек—машина" заключается в поддержании навыков оператора на требуемом уровне с помощью дополнительных тренировок.

За время господства классической парадигмы обеспечения эффективного человеко-машинного взаимодействия достаточно подробно разработаны прикладные аспекты психологического формирования эффективной человеко-машинной связи; решаются задачи профориентации, профотбора, профдифференциации, профподготовки, проектирования деятельности.

Изложенная выше схема проектирования разделяет задачи психологического проектирования интерфейса и его технической реализации, что на практике ведет к изоляции инженеров-проектировщиков от разработчиков человеческого компонента интерфейса и ведет к перманентному конфликту между психологами и инженерами в силу различия и несовместимости их понятийных областей. Проектирование интерфейса строится главным образом на опыте и интуиции инженера. Вследствие этого психология играет вспомогательную роль, ликвидируя методами отбора и обучения технические просчеты и ошибки, которые порождает человеческий фактор. Вместе с тем это широко используемая в инженерной практике технология создания эффективных интерфейсов для простых информационных систем.

Отметим, что классические инженерные представления об интерфейсе и работе человеческой психики основаны, главным образом, на здравом смысле и механистических моделях взаимодействий, что сильно обедняет проблемное поле интерфейсных систем.

Неклассическая методология проектирования

Появление новых методов проектирования интерфейсов обусловлено интенсивным развитием информационных технологий, ведущим к появлению сложных машин и механизмов, составляющих технические среды [13]. Наблюдается интенсивное внедрение тех-

нологий искусственного интеллекта во все сферы взаимодействий человека с искусственными средами. В таких условиях классический подход к проектированию интерфейсов работает плохо. Возможности чисто инженерного подхода к решению задач, возникающих при создании интерфейсов информационных систем, резко ограничены в силу большого числа возникающих при этом нетривиальных вариантов решений, отражающих взаимодействия с пользователем. Психологических знаний у инженеров-разработчиков и программистов для эффективного инженерного проектирования таких интерфейсов становится явно недостаточно. Появляется необходимость включения в эту работу специалистов в области инженерной психологии и эргономики.

В настоящее время задачи проектирования и оценки массовых пользовательских компьютерных интерфейсов решают в рамках прикладных дисциплин по учету человеческого фактора. Среди них наиболее известны в инженерной среде оценка и проектирование пользовательских качеств продукта (*Usability*) и использование при проектировании пользовательского опыта (*User Experiense*).

Однако следует признать, что рассматриваемые в данных направлениях свойства интерфейсов (полезность, юзабилити, доступность и привлекательность) отражают лишь некоторые статистические характеристики наблюдаемого пользовательского опыта без анализа его структуры и типов возникающих в системе отношений [14]. Проблема интерфейса в компьютерных системах и средах породила целый спектр новых проектировочных дисциплин, к которым следует отнести юзабилити-информатику (*Informatics Usability*), человеко-компьютерное взаимодействие (*Human-Computer Interaction*), эмоциональный дизайн (*Emotional design*). Однако все они строятся на прикладных исследованиях, не имея надлежащего научного базиса.

Информационные технологии предоставляют разработчикам эргатических систем широкий спектр средств, повышающих их интеллектуальность, формируя отношения человека-оператора с технической системой, аналогичные возникающим в условиях естественной социальной коммуникации.

Вместе с тем взаимодействие человека с искусственными информационными средами, наделенными искусственным интеллектом, отличается от его взаимодействия с естественными средами в силу дополнительности искусственных сред по отношению к когнитивному аппарату человека [15]. Возникает вопрос симбиоза между системами разной природы — биологическими, наделенными механизмами психического отражения и активного целеполагания, и техническими, реализующими алгоритмы и технологии искусственного интеллекта [16]. Переход человечества к новым формам технологического уклада, особенно к пятому и шестому, связанным с интенсивным развитием технологий *NBICS*-конвергенции и методов искусственного интеллекта, ведет к тотальной интеграции психофизиологической системы человека с техногенной средой [17], которая сопровождается появлением новых форм интерфейсных отношений.

Особое значение имеют техногенные среды эргатических систем, объединяющие большое число пользователей в рамках решения общих задач контроля и управления.

Системы интерфейса, объединяющие в единое целое интеллектуальную эргатическую систему и когнитивную систему оператора, отличаются от классических систем интерфейса, так как их функционирование носит коммуникативный, а не управляющий характер. Решение в системе принимается посредством коммуникативного акта, в результате которого субъектом принятия решения может стать система искусственного интеллекта, а не человек-оператор. Коммуникация рассматривается как социальная аутопоэтическая система, конституирующая социальные формы взаимодействий, в том числе и в эргатических системах с коллективным и групповым управлением [18].

Интерфейс в интеллектуальных эргатических системах должен создавать в операторе образ доверия к его "электронному партнеру", наделенному искусственным интеллектом, и обеспечивать эффективную коммуникацию. Эти задачи не могут быть решены в рамках классических системных представлений, в которых не учитывают процессы самоорганизации и эволюции возникающих системных объединений. Техногенные среды ведут к интеграции человека с машиной, формируя гибридные и симбиотические формы интеллектных образований организменного типа [19].

Постнеклассический подход

Постнеклассические подходы к проектированию интерфейса связаны с дальнейшим развитием системного подхода, введением новых форм системных взаимодействий, пронизывающих все формы отношений человека с миром, включающим осознание тотальной интеграции и межсвязности мира, ведущей к появлению гибридов природы и культуры, размытием границ между цифровым и материальным бытием [20]. В данном подходе используют парадигмы развивающихся, исторических систем. Сознание человека

рассматривают как интерфейс, связывающий субъекта с миром его действительности. Формирование сознания связано с созданием безопасной, комфортной, бесконфликтной картины мира, в которой субъект получает возможность свободно действовать, без опасений потерять свою субъектную индивидуальность и целостность организма. Интерфейс в постнеклассическом прочтении рассматривают как селективную границу между аутопоэтической системой человеческой психики, редуцирующей физическую реальность и пропускающей в конструирующую зону психики только неразрушающие формы описаний [21], и физической реальностью. Интерфейс при этом формирует не только неразрушающую/ориентирующую связь, но и самоорганизующуюся связь, конструирующую и конституирующую субъекта и мир его опыта.

Технологические уровни интерфейсной интеграции сложных систем

В настоящее время технологический прогресс человечества связан с процессами интенсивной конвергенцией ряда научных и технологических дисциплин (*NBICS*), с объединением их в научно-технологические комплексы, создающие базис для создания интерфейсов. В конечном итоге такие комплексы включают полную интеграцию человеко-машинной среды и человека на всех уровнях организации материи.

Развитие технологий позволяет создавать системы интерфейса, ориентированные не только на связь моторных и сенсорных компонентов тела человека с естественной средой, но и включающие все формы информационных отношений с искусственными средами, содержащими элементы дополненной и виртуальной реальностей [22]. Тотальная связность всех информационных процессов и ресурсов в глобальных сетях коммуникации и легкий доступ к ним позволяют создавать системы интерфейса с распределенным интеллектуальным содержанием, актуализирующиеся в процессе достижения цели эргатической системой. В зависимости от контекста деятельности это позволяет избирательно усиливать или компенсировать в случае недостаточности те или иные психические и интеллектуальные возможности оператора и машины.

Вместе с тем "технологическая свобода", возникшая в связи с развитием информационных технологий, может быть опасной при создании систем интерфейса сложных эргатических систем. Так, в исследованиях М. Б. Меликовой показано, что информационная техносреда, формируемая в современном самолете, ведет к ряду негативных последствий для летной деятельности:

- изменяется психологическая реальность полета, нарушаются психологические механизмы, ведущие к осмысленной деятельности пилота в полете;
- готовые к действию в информационной среде самолета "семантические импланты" тормозят собственную активность человека;
- невозможность проверки предлагаемого электронной системой содержания приводит к "слепому доверию" пилота технике;
- возникает "контаминация образа полета" — "взгляд на полет глазами инженера";
- развивается "энтропия образа полета" [23].

Введение различных методов интеграции пилота с автоматизированной системой информирования, таких как реализация метафоры "стеклянная кабина", создание "электронного помощника" и т. д. включают пилота в информационные взаимодействия с системами искусственного интеллекта самолета, негативно влияя на летное мастерство пилота.

Глобальная интеллектуальная информационная среда

Следующим уровнем интеграции человека и машины является создание "умного мира", в котором границы искусственной среды и человеческого тела и психики размываются. Техногенная среда становится тотальной формой интерфейса, обеспечивающего субъекту деятельности доступ ко всем формам создания, хранения и использования информации, энергии и пространства. Возникает целый ряд вопросов на этапе проектирования интерфейсов и учета взаимодействий в системе "оператор—эргатическая система—интеллектуальная информационная среда".

Интеграция различных технологий доступа, обработки, представления, хранения и использования информации с источниками информации, обладающими множественными входами-выходами и информационно-поисковыми системами в рамках единой глобальной информационной среды, ведет к появлению нового объекта технологии — *глобальной интеллектуальной информационно-коммуникационной среды*. Необходимо различать интерфейсы и свойства интеллектуальных информационных систем и интеллектуальных информационных сред (табл. 1).

Новые формы интеграции человека с информационным миром требуют усиления гуманитарного ком-

Таблица 1

Сравнение категорий "интеллектуальная информационная система" и "интеллектуальная информационная среда"

Критерии сравнения	Интеллектуальная информационная система	Интеллектуальная информационная среда
Тип образования	Программно-техническая система	Социотехническое коммуникативное единство аутопоэтического типа
Механизм создания	Локальное инженерное проектирование	Кооперативное взаимодействие, коммуникация, самоорганизация
Цель функционирования	Поддержка деятельности человека	Самовоспроизведение и эволюция
Механизм функционирования и эволюции	Отсутствует, поэтапное улучшение в процессе проектной деятельности	Непрерывное изменение в процессе самоорганизации и коэволюции человеческого общества
Входы	Единичные	Множественные, по числу пользователей
Интерфейсы	Создаются под целевую задачу системы, уникальны, связаны с оптимизацией системы по критериям эффективности	Формируются как элементы техногенной культуры, возникающей в среде, действует тенденция к унификации в рамках пользовательского опыта
Вид аккумуляции опыта. Источник опыта	Локальный, разработчиков системы в процессе ее проектирования	Циклический, рекурсивный, избирательный, пользователей терминалов среды и культурных сообществ

понента инженерного знания и перехода к методам постнеклассической эргономики [24]. Необходимо различать свойства симбионтных объединений (интеллектные симбионты), возникающих в организованных искусственных средах. Симбионтными отношениями будем называть отношения, возникающие между человеком и технической системой (техногенной средой), наделенной искусственным интеллектом. Они отличаются от симбиотических отношений, возникающих в живой природе, тем, что в них отражена ведущая роль человека в процессах конструирования симбиотных образований. Кроме того в симбиозах рассматривается взаимная выгода объединяющихся систем, а в симбионтах — выгода по отношению к человеку.

Симбиотические отношения в интеллектуальных системах и средах

Интеллектуальные системы и среды представляют собой *системы организованной сложности*. Будучи организованными человеком, они содержат в себе в скрытой форме часть функций, присущих системам, наделенным естественным интеллектом. Это искусственный интеллект, воплощенный в искусственную среду. Основные виды интеллектных симбионтов приведены в табл. 2.

Интерфейс в системных моделях когнитивной науки

Проблема создания эффективного интерфейса в сложных эргатических системах и средах включает междисциплинарные технические и гуманитарные аспекты, отражающие различные формы объединений человека с техникой. Можно дать ряд общих определений интерфейса, в которых сделан акцент на свойствах системных образований, обеспечивающих активную и эффективную деятельность человека в эргатических системах и средах техногенного мира. В них учитываются неклассические представления когнитивных наук, рассматривающие человека как сложную систему аутопоэтического типа (В. И. Аршинов, В. Г. Буданов, Ф. Варела, Н. Луман, У. Матурана, Х. фон Ферстер, и др. [25—28]).

В соответствии с этими определениями интерфейс представляет собой:

- неразрушающую опосредованную межсистемную связь;
- средство опосредованного достижения цели;
- механизм и среду интеграции систем;
- средство погружения в среду;
- средство взаимной ориентации систем;
- средство обеспечения границы различений между системами;
- пограничную среду, передающую селективное, неразрушающее действие;

Таблица 2

Виды интеллектных симбионтов

Тип интеллектного образования	Отношения между компонентами	Центр активности и управления, механизм	Отношения со средой деятельности, границы
Естественный интеллект	Самоорганизация, аутопоэтическая система	Сознание, эго-система человека	Активное преобразование мира, границы динамично изменяются
Искусственный интеллект	Фиксированная или переменная программно-аппаратная структура	Программа, алгоритм в структурированной или структурируемой среде	Реализация алгоритма, ситуативное управление, границы фиксированные
Гибридный интеллект	Симбиоз, адаптация организованного и аутопоэтического компонентов к среде, объединения на макроуровне при приоритете сознания	Человек в структурированной среде, кооперативное взаимодействие человека и машины	Взаимная адаптация естественного и искусственного интеллектов, границы переменные
Диффузный интеллект	Селективные связи на всех уровнях аутопоэтически организованной и организуемой среды и человека	Возникает в организованной среде, сетевые взаимодействия искусственных и естественных сред	Синергетическое объединение, границы формируются под задачу

- правила межсистемных взаимодействий;
- средство межсистемной коммуникации.

Заключение

Проблема интерфейса является центральной при создании дружественной пользователю информационных систем глобальной техногенной среды и эргатических систем, усиливающих и увеличивающих возможности человечества, обеспечивающие его безопасность и эффективность при реализации трудовой деятельности. Для ее решения недостаточно технологий инженерного проектирования. Требуется междисциплинарный подход, включающий новые технологии учета психологических знаний и системных представлений о живой природе и социальных системах.

Список литературы

1. **Сергеев С. Ф., Падерно П. И., Назаренко Н. А.** Введение в проектирование интеллектуальных интерфейсов. СПб.: СПБГУ ИТМО, 2011. 108 с.

2. **Галактионов А. И.** Представление информации оператору (исследование деятельности человека-оператора производственных процессов). М.: Энергия, 1969. 136 с.

3. **Сергеев С. Ф.** Инженерная психология и эргономика. М.: НИИ школьных технологий, 2008. 176 с.

4. **Раскин Д.** Интерфейс: новые направления в проектировании компьютерных систем. СПб.: Символ-Плюс, 2007. 272 с.

5. **Сергеев С. Ф.** Инженерно-психологическое проектирование сложных эрготехнических сред: методология и технологии // Актуальные проблемы психологии труда, инженерной психологии и эргономики / Под ред. В. А. Бодрова, А. Л. Журавлева. Вып. 1. М.: Изд-во Института психологии РАН, 2009. С. 429—449.

6. **Сергеев С. Ф.** Эргономика иммерсивных сред: методология, теория, практика: дис. ... д-ра психол. наук: 19.00.03. СПб., 2010. 420 с.

7. **Сергеев С. Ф.** Регуляция, саморегуляция, самоорганизация, саморазвитие в понятийном базисе психологии // Актуальные проблемы психологии труда, инженерной психологии и эргономики. Выпуск 4 / Под ред. В. А. Бодрова, А. Л. Журавлева. М.: Изд-во Института психологии РАН, 2012. С. 238—259.

8. **Справочник** по инженерной психологии / Под ред. Б. Ф. Ломова. М.: Машиностроение, 1982. 368 с.

9. **Ломов Б. Ф.** Человек и техника. Очерки инженерной психологии. М.: Советское радио, 1966. 464 с.

10. **Завалова Н. Д., Ломов Б. Ф., Пономаренко В. А.** Принцип активного оператора и распределение функций между человеком и автоматом // Вопросы психологии. 1971. № 3. С. 3—12.

11. **Сергеев С. Ф.** Методология проектирования тренажеров с иммерсивными обучающими средами // Научно-технический вестник СПбГУ ИТМО. 2011. № 1 (71). С. 109—114.

12. **Сергеев С. Ф.** Обучающая среда: концептуальный анализ // Школьные технологии. 2006. № 5. С. 29—34.

13. **Сергеев С. Ф.** Глобальные техногенные среды в эволюции человеческой цивилизации // Вестник Московского университета им. С. Ю. Витте. Серия 1: Экономика и управление. 2013. № 1. С. 80—86.

14. **Речинский А. В., Сергеев С. Ф.** Разработка пользовательских интерфейсов. Юзабилити-тестирование интерфейсов информационных систем: учеб. пособие. СПб.: Изд-во Политехнического университета, 2012. 145 с.

15. **Сергеев С. Ф.** Методология создания мехатронных систем с искусственным интеллектом // Известия ТулГУ. Технические науки: В трех частях. Ч. 1. Тула: Изд-во ТулГУ, 2011. Вып. 5. С. 245—249.

16. **Сергеев С. Ф.** Интеллектные симбионты в эргатических системах // Научно-технический вестник информационных технологий, механики и оптики. 2013. № 2 (84). С. 149—154.

17. **Сергеев С. Ф.** Рефлексивная автоэволюция глобальных интеллектных техногенных сред // Рефлексивные процессы и управление. Сборник материалов IX Международного симпозиума. 17—18 октября 2013 г., Москва / Отв. ред. В. Е. Лепский. М.: Когито-Центр, 2013. С. 245—248.

18. **Луман Н.** Общество как социальная система. Пер. с нем./ А. Антоновский. М.: Логос, 2004. 232 с.

19. **Сергеев С. Ф.** Проблема эффективного взаимодействия человека-оператора с интеллектуальными техническими системами и средами // Материалы 3-го междунар. науч.-техн. семинара "Современные проблемы прикладной математики, информатики, автоматизации, управления". 9—13 сентября 2013 г., г. Севастополь. М.: ИПИ РАН, 2013. С. 183—197.

20. **Сергеев С. Ф.** Наука и технология XXI века. Коммуникации и НБИКС-конвергенция // Глобальное будущее 2045. Конвергентные технологии (НБИКС) и трансгуманистическая эволюция / Под ред. проф. Д. И. Дубровского. М.: МБА, 2013. С. 158—168.

21. **Сергеев С. Ф.** Проблема редукции в когнитивном механизме сознания // Проблема сознания в междисциплинарной перспективе / Под ред. В. А. Лекторского. М.: Канон+ РООИ Реабилитация, 2014. С. 245—254.

22. **Сергеев С. Ф.** Обучающие и профессиональные иммерсивные среды. М.: Народное образование, 2009. 432 с.

23. **Меликова М. Б.** Социокультурный аспект интеграции перспективных систем "летчик—самолет" // Материалы XLVII Научных чтений памяти К. Э. Циолковского. Калуга, 2012. С. 232—233.

24. **Сергеев С. Ф., Захаревич А. П.** Постклассическая эргономика сложных сред: базовые понятия // Материалы 7-й научно-технической конференции "Мехатроника, автоматизация, управление". СПб.: ЦНИИ Электроприбор, 2010. С. 357—360.

25. **Аршинов В. И., Буданов В. Г.** Синергетика как методология коммуникативного конструктивизма // Конструктивистский подход в эпистемологии и науках о человеке / Отв. ред. акад. В. А. Лекторский. М.: Канон+, РООИ Реабилитация, 2009. С. 241—274.

26. **Матурана У., Варела Ф**. Древо познания. М.: Прогресс-Традиция, 2001. 224 с.

27. **Луман Н**. Социальные системы. Очерк общей теории. СПб.: Наука, 2007. 645 с.

28. **Heinz von Foerster.** Understanding Understanding: Essays on Cybernetics and Cognition, New York: Springer-Verlag, 2003. 362 p.

ИНФОРМАЦИЯ

23—24 октября 2014 г. в Москве, в центре Digital October пройдет Десятая юбилейная конференция

«Разработка ПО / CEE-SECR 2014»

За годы существования SECR заслуженно стал одним из важнейших событий ИТ-индустрии. Более 900 специалистов соберутся на конференцию в этом году: от программистов и представителей академического сообщества до предпринимателей и инвесторов. Участников ждут выступления экспертов мирового уровня, мастер-классы, дискуссии, общение со спикерами и многое другое.

Во время конференции выступления пройдут одновременно в несколько потоков, что даст участникам возможность выбирать наиболее интересную для себя тему в каждый момент.

Приглашаем Вас:

- Подать доклад – поделитесь своим уникальным опытом с коллегами и выступите на одной сцене с признанными лидерами мировой и российской индустрии разработки ПО. Авторы принятых работ участвуют в конференции бесплатно.
- Принять участие – этот SECR станет юбилейным, а значит наиболее ярким и интересным за историю конференции. Успейте зарегистрироваться со скидкой! Действуют специальные предложения группам, студентам.

Более подробная информация на сайте ***www.secr.ru***

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4

Дизайнер *Т. Н. Погорелова*. Технический редактор *Е. М. Патрушева*. Корректор *Т. В. Пчелкина*

Сдано в набор 05.02.2014 г. Подписано в печать 20.03.2014 г. Формат 60×88 1/8. Заказ PI414
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1.