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 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.

УДК 004.052.32

Ю. П. Кораблин, д-р техн. наук, проф., Российский государственный социальный университет, e-mail: y.p.k@mail.ru
Е. Г. Павлов, аспирант, Российский государственный социальный университет, программист, Исследовательский центр Samsung, e-mail: lucenticus@gmail.com

Разработка верификатора объектов синхронизации драйверов для операционной системы Linux на основе процессной семантики

Описывается инструментальное средство для поиска ошибок в механизмах синхронизации драйверов операционной системы Linux, которое использует метод их обнаружения на основе семантических моделей. Рассматриваемое инструментальное средство позволяет обнаруживать такие ошибки синхронизации как зацикливания, взаимоблокировки, двойные блокировки. Описана процессная семантика объектов синхронизации ядра Linux на языке асинхронных функциональных схем.

Ключевые слова: *верификация, драйверы устройств, взаимодействующие последовательные процессы, эквациональная характеристика, алгебраическая семантика*

Введение

Операционные системы являются одним из самых важных компонентов вычислительной системы. Однако как и любое программное обеспечение, операционная система может содержать ошибки. Современные системы имеют довольно сложную структуру и состоят из множества взаимосвязанных между собой модулей.

Драйвер — программа, которая обеспечивает взаимодействие между устройством и операционной системой. В большинстве случаев драйвер входит в состав операционной системы, так как ему необходимо получать прямой доступ к привилегированным функциям ядра. Для каждого устройства необходим отвечающий его специфике драйвер, поэтому набор таких драйверов является самой объемной частью операционной системы по числу строк кода. Подавляющее число драйверов разрабатывается фирмой-производителем устройства, что зачастую негативно сказывается на качестве драйвера. Поскольку драйвер является частью операционной системы, то ошибка в нем может привести к сбою всей системы.

Как показано в работе [1], одной из самых распространенных ошибок в драйверах Linux являются ошибки в реализации механизмов синхронизации. Они возникают при неправильном использовании объектов синхронизации (семафоров, мьютексов, спин-блокировок и др.), а также при наличии разделяемых ресурсов без ограничения доступа к ним с помощью объектов синхронизации. Отмеченные причины приводят к появлению в драйверах взаимоблокировок, двойных блокировок и состояний гонок за ресурсы (*race condition*). Такие ошибки довольно сложно обнаружить и воспроизвести, так как они могут проявляться лишь при определенных взаимодействиях между функциями драйвера, а также при взаимодействии нескольких драйверов, имеющих общие ресурсы.

В настоящей статье описывается инструментарий для поиска ошибок в программной реализации механизмов синхронизации в драйверах Linux. В основе его лежит метод анализа процессной семантики программы, разработанный авторами [2]. Данное инструментальное средство позволяет обнаруживать такие ошибки синхронизации как зацикливания, взаимоблокировки и двойные блокировки.

Инструментарий для верификации драйверов операционных систем

Большинство средств анализа драйверов на настоящее время построено на основе метода проверки моделей (*model checking*) [3]. Это связано, прежде всего, с хорошей проработанностью данного метода, а также с наличием инструментальных средств для верификации, таких как Spin [4], BLAST [5], CBMC [6] и др. Рассмотрим наиболее распространенные из них.

Для верификации драйверов операционных систем семейства Windows компания Microsoft разработала программный пакет под названием Static Driver Verifier, поставляемый с 2006 г. как часть Microsoft Windows Driver Development Kit. Данный верификатор основан на технологии SLAM, разработанный в Microsoft Research [7].

Инструментарий для статического анализа Static Driver Verifier (далее SDV) предназначен для автоматической проверки во время компиляции кода драйвера Windows в целях выявления нарушений правил драйверной модели. Средство SDV исследует ветви исполнения кода драйвера устройства, символически исполняя его исходный код на своей собственной модели операционной системы. Когда SDV выполняет проверку драйвера, он объединяет его исходный код с моделью операционной системы и вводит этот комбинированный код на языке C в верификационную машину SLAM. Она оснащает этот код правилами, выбранными при запуске процесса верификации, а затем выполняет всестороннюю проверку этого кода методом *model checking*. По результатам проведенного анализа генерируются статистические данные о запуске и детальные трассы ошибок, для исследования которых разработаны специализированные графические средства, которые входят в состав Driver Development Kit.

Авторы работы [8], основываясь на технологии Microsoft, разработали верификатор драйверов Linux. Они расширили язык спецификаций SLIC и вместо SLAM использовали верификатор на основе проверки моделей CBMC [6]. На базе документации ядра были сформулированы некоторые правила для драйверов Linux, реализованные на языке SLICx. Таким образом, этот инструментарий позволил использовать стиль SDV для верификации драйверов Linux. Это средство выполнено в виде плагина для среды разработки Eclipse.

Иной подход реализован в предикативном верификаторе DDVerify [9]. Он является полностью автоматизированным инструментальным средством проверки кода, который, учитывая исходный код Linux-драйвера устройства, генерирует соответствующую обертку для драйвера и с использованием SATABS [10] проверяет, где драйвер нарушает условия модели ядра. Правила корректности задаются как часть модели ядра. Код ограничений, накладываемых правилами, задается вместе с кодом, описывающим семантику функции. В данном подходе можно проверять только те функции, которые привязываются к драйверу с помощью линковки. По этой причине для использования DDVerify требуется существенно изменять заголовочные файлы ядра.

В 2009 г. на базе Института системного программирования Российской академии наук был создан Центр верификации операционной системы Linux. Основным средством для верификации в Центре является инструментарий Linux Driver Verification (далее LDV) [11].

Процесс верификации драйвера в LDV начинается с инициализации системы LDV-Core. Перед началом работы на основе архива с исходным кодом ядра Linux создается копия этого ядра со специально модифицированной подсистемой сборки. Далее LDV-Core запускает процесс компиляции драйверов, и одновременно с этим читается поток команд компиляции и линковки, при этом из них выделяются те, которые относятся к верифицируемым драйверам. Затем создается для каждого типа драйвера одна или несколько моделей окружения и добавляется код моделей и проверок указанных правил корректности к соответствующим файлам драйвера. Далее код отправляется на вход верификатору (BLAST, CPAchecker и др.), который уже не имеет представления о том, что поданый ему на вход код является драйвером ядра операционной системы Linux. На настоящее время в LDV используются только верификаторы достижимости. Этот факт означает, что инструментарий предназначен лишь для выявления нарушений правил корректности.

Верификация на основе семантических моделей

В работе [2] для доказательства свойств программ предлагается использовать семантические модели. Семантика языков распределенного программирования исследуется посредством постановки в соответствие программам множества вычислительных последовательностей и анализа семантических значений в заданной алгебраической модели вычислительных последовательностей.

Методика исследования семантики [12] программ разработана для модели языка распределенного программирования L и расширена для языка асинхронных функциональных схем (АФС). Для анализа свойств программы сначала необходимо преобразовать ее в соответствующую программу на языке L или АФС.

Рассмотрим более подробно язык АФС, неформальное описание которого представлено в работе [12]. Пусть заданными являются следующие синтаксические области:

Com — множество элементарных команд с типичным элементом *a*;

Exp — множество булевских выражений с типичным элементом *b*;

BConst — множество булевских констант, содержащее константы *tt* (тождественно истинное значение) и *ff* (тождественно ложное значение);

RCom — множество команд ввода с типичным элементом *read*;

WCom — множество команд вывода с типичным элементом *write*;

CPLab — множество меток каналов и процессов с типичными элементами *i* и *j*;

CNom — множество номеров входов/выходов каналов с типичными элементами *k* и *l*;

Cmd — множество команд с типичным элементом *c*;

CWait — множество команд ожидания с типичным элементом *wait*;

TExp — множество временных выражений с типичным элементом *time*.

Множество АФС-программ *Prog* с типичным элементом *pr* определяется следующим образом:

$pr ::= NET\ \{can\}\ BEGIN\ fproc\ END$

$can ::= CHAN\ j::type(k):type(l) \mid can_1;\ can_2$

$type ::= ALL \mid ANY$

$fproc ::= FUN\ i::c \mid fproc_1;\ fproc_2$

$c ::= a \mid skip \mid exit \mid break \mid wait(time) \mid read(i,\ l) \mid write(i,\ k) \mid SEQ(c\ \{,\ c\}) \mid$

$PAR(c\ \{,\ c\}) \mid ALT(gc) \mid LOOP(ALT(gc))$

$gc ::= g \to c \mid gc\ \{;\ gc\}$

$g ::= tt \mid ff \mid b \mid wait(time) \mid read(i,\ l) \mid write(i,\ k)$

Определим семантические области, на которые отображаются значения АФС-программ. Предполагаем, что заданными являются следующие семантические области:

- множество значений элементарных команд *ACom* с типичным элементом *A*;
- множество значений булевских выражений *BExp* с типичным элементом *B*;
- множество значений команд ввода *PICom* с типичным элементом *IN*;
- множество значений команд вывода *POCom* с типичным элементом *OUT*;
- множество взаимодействий *PCom* с типичным элементом γ;
- множество *Const*, содержащее константы τ (тождественное преобразование), $\varnothing$ (останов), *T* (тождественно-истинное значение) и *F* (тождественно-ложное значение);
- элементы *BREAK*, *EXIT* и *TIME*, являющиеся значениями команд завершения цикла, завершения процесса и ожидания соответственно.

Определим множество тестов *TEST* с типичным элементом β и множество действий *ACTION* с типичным элементом α следующим образом:

$\beta ::= T \mid F \mid B \mid TIME \mid IN \mid OUT \mid \gamma;$

$\alpha ::= \tau \mid \varnothing \mid A \mid IN \mid OUT \mid \gamma \mid TIME \mid BREAK \mid EXIT$

Определим множество вычислительных последовательностей (ВП) *CPath* с типичным элементом *cp*.

$cp ::= \alpha \mid \beta\ \hat{}\ cp \mid cp_1 \circ cp_2.$

Здесь "^" и "∘" — операции последовательной композиции ВП.

Через *SP* с типичным элементом *sp* обозначим множество всех подмножеств *CPath*. Определим на множестве *SP* операции теоретико-множественного объединения ($sp_1 + sp_2$), последовательной композиции ($sp_1 \circ sp_2$), параллельной композиции ($sp_1 \parallel sp_2$), теоретико-множественного вычитания ($sp_1 \backslash sp_2$) и минимальной фиксированной точки (sp^+).

Зададим семантическую функцию для функциональных процессов $\mathbb{F}: Fproc \to SP$, где *FProc* — множество функциональных процессов с типичным элементом *fproc*.

$\mathbb{F}[fproc_1;\ fproc_2] = \mathbb{F}[fproc_1] \parallel \mathbb{F}[fproc_2],$

$\mathbb{F}[FUN\ i :: c] = \mathbb{C}[c],$

где $\mathbb{C}: Cmd \to SP$ есть функция, ставящая в соответствие командам языка АФС множества ВП. Функция $\mathbb{C}$ для языка АФС задается следующим образом:

$\mathbb{C}[a] = A;$

$\mathbb{C}[skip] = \tau;$

$\mathbb{C}[exit] = EXIT;$

$\mathbb{C}[break] = BREAK;$

$\mathbb{C}[wait(time)] = TIME;$

$\mathbb{C}[read(i,\ l)] = IN_{i,\ l};$

$\mathbb{C}[write(i,\ k)] = OUT_{i,\ k};$

$\mathbb{C}[SEQ(c_1;\ c_2)] = \mathbb{C}[c_1] \circ \mathbb{C}[c_2];$

$\mathbb{C}[ALT(gc_1;\ gc_2)] = \mathbb{C}[gc_1] + \mathbb{C}[gc_2];$

$\mathbb{C}[tt \to c] = \mathbb{E}[tt]\ \hat{}\ \mathbb{C}[c];$

$\mathbb{C}[ff \to c] = \mathbb{E}[ff]\ \hat{}\ \mathbb{C}[c];$

$\mathbb{C}[b \to c] = \mathbb{E}[b]\ \hat{}\ \mathbb{C}[c];$

$\mathbb{C}[wait(time) \to c] = \mathbb{C}[wait(time)\ \hat{}\ \mathbb{C}[c];$

$\mathbb{C}[read(i,\ l) \to c] = \mathbb{C}[read(i,\ l)\ \hat{}\ \mathbb{C}[c];$

$\mathbb{C}[write(i,\ k) \to c] = \mathbb{C}[write(i,\ k)\ \hat{}\ \mathbb{C}[c];$

$\mathbb{C}[LOOP(ALT(gc))] = (\mathbb{C}[ALT(gc)])^+.$

Определим семантическую функцию для выражений:

$\mathbb{E}: Exp1 \to TEST;$

$\mathbb{E}[tt] = T;$

$\mathbb{E}[ff] = F;$

$\mathbb{E}[b] = B.$

Канал связи интерпретируется как циклический процесс, который вначале принимает данные от функциональных процессов, а затем, когда все необходимые данные от функциональных процессов (в соответствии с логикой входов этого канала) уже приняты, передает данные функциональным процессам (в соответствии с логикой выходов этого канала).

Алгоритм верификации на основе семантических моделей выглядит следующим образом. Вначале для программы на исходном языке программирования строится соответствующая ей программа на языке АФС. Для АФС-программы строится семантическое значение, вначале задаются априорные семантики каждого

функционального процесса и канала связи программы, затем они объединяются посредством операции композиции в алгебре вычислительных последовательностей. Далее на основе семантического значения программы осуществляется построение системы рекурсивных уравнений, опираясь на которую проводится поиск ошибочных состояний (зацикливаний, блокировок и др.). Более подробно процесс верификации описан в работе [2].

Рассмотрим основные семантические правила преобразования объектов синхронизации драйверов Linux в язык АФС. Сами объекты синхронизации ядра Linux описаны в работе [1].

Операционная семантика условных переменных. Условные переменные позволяют синхронизировать задания, при этом один или несколько процессов ожидают на условной переменной, пока другой процесс не отправит им сигнал о завершении события.

Семантическая функция для *wait_for_completion*(*op*), которая ожидает сигнала о завершении от переменной *op* типа *struct completion,* задается следующим образом:

$C[wait\_for\_completion(op)] = read(op, 1).$

Здесь и далее в статье $C[c]$ задает семантическую функцию, которая ставит в соответствие элементам синхронизации ядра Linux команды на языке АФС.

Для операции *complete*(*op*), которая посылает сигнал о завершении операции *op* всем ожидающим процессам, семантическая функция имеет вид:

$C[complete(op)] = write(op, 1).$

Операционная семантика спин-блокировок. Спин-блокировки в драйверах встречаются довольно часто. Семантически спин-блокировка представляет собой циклическую проверку возможности захвата объекта типа *spinlock_t.* На языке АФС в качестве объекта для захвата выступает канал вида $ALL(1):ALL(1)$. Захват блокировки соответствует попытке записи в канал. Данная попытка повторяется до успешной записи, что реализуется посредством цикла *LOOP.* Семантическая функция для функции захвата блокировки выглядит следующим образом:

$C[spin\_lock(s\_lock)] = LOOP(ALT(write(s\_lock, 1)) \rightarrow break)).$

Для освобождения спин-блокировки необходимо считать данные из канала, т.е. функции *spin_unlock*() соответствует операция $read(i, 1)$:

$C[spin\_unlock(s\_lock)] = read(s\_lock, 1).$

Функции попытки захвата блокировки *spin_trylock*() соответствует семантическая функция:

$C[spin\_trylock(lock)] = ALT(write(lock, 1) \rightarrow skip; tt \rightarrow skip).$

Операционная семантика спин-блокировок чтения-записи. Спин-блокировка чтения-записи реализуется на языке АФС посредством двух каналов для чтения и записи соответственно. Если захватывается блокировка на чтение, то процесс осуществляет запись в канал для чтения, если для записи — то процесс записывает в канал для записи. В целом, семантика спин-блокировок чтения-записи очень схожа с простыми спин-блокировками, лишь добавляются дополнительные проверки при захвате блокировки.

Семантические функции для захвата и освобождения спин-блокировки чтения-записи имеют следующий вид:

$C[read\_lock(rwlock)] = LOOP(ALT(write(r, 1) \rightarrow break)); read(r, 1); LOOP(ALT(write(w, 1) \rightarrow break));$

$C[read\_unlock(rwlock)] = read(r, 1);$

$C[write\_lock(rwlock)] = LOOP(ALT(write(w, 1) \rightarrow break)); read(w, 1); ALT(write(r, 1) \rightarrow skip) \ldots read(r, 1) \ldots$

$C[write\_unlock(rwlock)] = read(w, 1).$

Здесь w и r — каналы, соответствующие захвату спин-блокировки на запись и чтение соответственно.

Операционная семантика семафоров. Семафоры на языке АФС моделируются несколько сложнее. Семафор ограничивает число потоков в критическом участке кода специальным счетчиком, который моделируется на языке АФС каналами (значение счетчика должно совпадать с числом каналов). Таким образом, выполнение операции *down*(*sem*) на АФС соответствует последовательной проверке возможности записи в первый канал. Если этот канал занят, то осуществляется попытка записать во второй канал и т. д. Если свободный канал не найден, то выполнение прерывается в ожидании освобождения канала. При выполнении операции *up*(*sem*) осуществляется обратный порядок действий: последовательно происходит попытка считывания из канала N, затем $N-1$ и так далее до канала 1, где N — максимальное значение счетчика семафора. Таким образом, семантические функции для захвата и освобождения семафора имеют вид:

$C[down(sem)] = ALT(write(1, 1) \rightarrow skip; \ldots; write(N, 1) \rightarrow skip)$

$C[up(sem)] = ALT(read(N, 1) \rightarrow skip; \ldots; read(1, 1) \rightarrow skip).$

Операционная семантика семафоров чтения-записи. Семафоры чтения-записи реализуются аналогично спин-блокировкам чтения-записи за исключением того, что ожидающий процесс приостанавливается, а не входит в цикл в ожидании открытия семафора. В семафорах чтения-записи счетчик потоков всегда равен единице. Семантические функции для захвата и освобождения семафоров чтения-записи относительно простые по сравнению со спин-блокировками чтения-записи:

$C[down\_write(rwsem)] = write(r, 1); write(w, 1); read(r, 1);$

$C[up\_write(rwsem)] = read(w, 1);$

$C[down\_read(rwsem)] = write(w, 1); write(r, 1); read(w, 1);$

$C[up\_read(rwsem)] = read(r, 1).$

Как видно из семантической функции, в языке АФС добавлены дополнительные проверки возможности

записи в канал *w* и *r* при захвате семафора на чтение и запись соответственно, что аналогично захвату соответствующей спин-блокировки чтения-записи.

Операционная семантика мьютексов. Мьютексы по сути являются семафорами со счетчиком, равным единице. Они используются там, где в критический участок кода должен зайти только один поток. Семантически мьютексы отличаются от спин-блокировок тем, что при неудачном захвате мьютекса процесс засыпает. На языке АФС операция захвата мьютекса соответствует попытке записи в канал. Канал устроен таким образом, что если запись невозможна в данный момент, то процесс засыпает в ожидании освобождении канала. Для освобождения мьютекса необходимо прочитать данные из канала. Семантические функции для мьютексов задаются следующим образом:

$C[down(mutex)] = write(mutex, 1);$

$C[up(mutex)] = read(mutex, 1).$

Основные компоненты инструментария верификации

Верификатор объектов синхронизации имеет модульную структуру. Отдельные компоненты связаны между собой последовательно, а именно выход одного компонента передается на вход следующего. Этот подход позволяет модифицировать отдельные части верификатора для адаптации к иному типу программ. Верификатор состоит из следующих основных частей:

- препроцессор;
- транслятор кода драйвера в программу на языке АФС;
- модуль представления программы АФС в виде системы рекурсивных уравнений;
- анализатор системы рекурсивных уравнений.

Далее рассмотрим особенности реализации каждого компонента.

Препроцессор. Данный препроцессор реализован в виде сценария на языке bash. В его обязанности входит подготовка кода драйвера к трансляции в язык АФС.

Исходный код драйвера поступает на вход препроцессора, который считывает файл Makefile драйвера и анализирует зависимости между файлами исходного кода модуля. Затем для каждого файла на языке С препроцессор осуществляет вызов компилятора GCC с ключом -E:, эта команда запускает препроцессор языка С, который обрабатывает все операторы препроцессора языка С (#include, #define и др.). При необходимости подключения какого-либо заголовочного файла препроцессор может искать недостающие файлы в директориях, которые указаны в его параметрах. На выходе препроцессора получается один файл исходного кода драйвера на языке С. Этот файл передается на вход транслятора в язык АФС.

Транслятор кода драйвера в язык АФС. После всех подготовительных процедур препроцессор передает управление транслятору в язык АФС. В данном случае АФС выбран как основной язык для промежуточного представления структуры драйвера. Выбор этого языка обусловлен удобством при преобразовании в систему рекурсивных уравнений и наличием уже заданных в работе [12] семантических областей для данного языка. Транслятор из языка С в язык АФС будем называть front-end, этот термин часто используется для трансляторов из исходного языка (в данном случае это язык С) в язык промежуточного представления (язык АФС).

Транслятор написан с использованием генераторов лексических анализаторов GNU Flex и синтаксических анализаторов GNU Bison.

На вход Flex подается файл на языке Lex и описывает генерируемый лексический анализатор. Компилятор Flex преобразует этот файл в программу на языке С. Эта программа представляет собой работающий лексический анализатор, который может получать на вход поток символов и выдавать поток токенов. Правила трансляции имеют следующий вид:

Шаблон {Действие}.

Каждый шаблон является регулярным выражением. Действия представляют собой фрагменты кода, написанные на языке С. Сгенерированный лексический анализатор в дальнейшем используется для генерации синтаксического анализатора.

Генератор синтаксических анализаторов Bison на вход получает файл со спецификацией разрабатываемого транслятора на языке Yacc, который он преобразует в синтаксический анализатор на языке С. Основой файла на языке Yacc являются правила трансляции грамматических выражений и семантических действий для каждого из этих правил. На выходе Bison получается программа-транслятор, которая на основе заданных семантических правил проводит преобразование исходной программы.

В основе транслятора кода драйвера в язык АФС лежит реализация шаблона для анализатора языка ANSI C, а также ранние версии компилятора GCC, синтаксис которого был задан с использованием Bison. Шаблон синтаксического анализатора был дополнен правилами для генерации абстрактного синтаксического дерева (АСД), помимо этого добавлено распознавание GCC-расширений языка С (атрибуты, ассемблерные вставки, инициализация структур и др.). После лексического и синтаксического анализа кода драйвера на выходе получается АСД программы. Далее это дерево используется для анализа структуры программы и для генерации программы на языке АФС.

Вначале транслятор ищет функцию *module_init*() и считывает название функции инициализации. Далее анализатор просматривает тело функции инициализации и находит инициализацию устройства при помощи функции *cdev_init,* из этого вызова считывается идентификатор структуры *file_operations.* Эта структура содержит указатели на операции *open, close, read, write* и др. Именно данные операции и представляют особый интерес, так как они зачастую выполняются параллельно. Каждой операции драйвера соответствует функциональный процесс на языке АФС. Транслятор исследует каждую из этих функций и собирает информацию об использовании в них объектов синхронизации. Ли-

нейные операторы не представляют интереса и поэтому заменяются на команды вида C_i. Далее происходит анализ использования данных объектов между функциями драйвера. Эти взаимосвязи и используются для генерации программы на языке АФС. На выходе после выполнения всех шагов front-end записывает программу на языке АФС в файл с расширением .afs. Данный файл используется для дальнейшего анализа программы.

Преобразование АФС-программы в систему рекурсивных уравнений. Следующий этап после представления драйвера на языке АФС — это преобразование программы на языке АФС в систему рекурсивных уравнений.

Сначала на основе заданных в предыдущем разделе семантических функций $\mathbb{F}$ и $\mathbb{C}$ для каждого процесса определяется априорная семантика. Программно это реализовано с помощью модуля, который проводит нисходящий анализ АФС-программы методом рекурсивного спуска и для каждой команды на языке АФС применяет семантические функции $\mathbb{F}$ и $\mathbb{C}$. После прохода полученные априорные семантики процессов и каналов связи хранятся в виде абстрактного синтаксического дерева.

Далее полученные априорные семантики объединяются посредством операции параллельной композиции в алгебре вычислительных последовательностей. Это объединение реализовано на основе определения операции параллельной композиции $\|$, заданной в работе [12]. Применение этого определения, а также последовательное использование свойств операций "^", "$\circ$" и "+" позволяют получить первое уравнение системы. Таким образом, на основе априорной семантики процессов строится единое АСД для всей программы целиком. Все дальнейшие операции проводятся над этим деревом.

После получения первого уравнения системы вызывается функция, которая ищет операции последовательной композиции "^" и "$\circ$" в программе и считывает операнды данной операции. Затем каждый правый операнд обозначается как новое уравнение системы и все поддерево, соответствующее правому операнду, сохраняется как новое уравнение. Далее для каждого из этих поддеревьев вызываются те же функции, которые вызывались для первого уравнения. Данная операция рекурсивно выполняется и для каждого из полученных уравнений до тех пор, пока в полученных операциях присутствуют операции последовательной композиции.

Например, пусть у нас получилось следующее первое уравнение:

$$P_1 = T \wedge A + OUT_{1,1} \circ IN_{1,2}.$$

Заменяя правые операнды операций "^" и "$\circ$" на P_2 и P_3, получаем:

$$P_1 = T \wedge P_2 + OUT_{1,1} \circ P_3;$$

$$P_2 = A;$$

$$P_3 = IN_{1,2}.$$

Конечность данной системы рекурсивных уравнений доказывается в работе [12]. Полученная система задает множество вычислительных последовательностей, сопоставляемое структуре драйвера. На выходе система рекурсивных уравнений записывается в файл с расширением .sem.

Анализатор системы рекурсивных уравнений. Сама по себе система рекурсивных уравнений довольно сложна для анализа программы. Поэтому в целях упрощения обнаружения исключительных ситуаций реализован инструмент для анализа системы рекурсивных уравнений. Он позволяет анализировать систему рекурсивных уравнений в файле с расширением .sem и выводить в удобочитаемом виде информацию о возможных ошибках синхронизации в драйвере.

Анализатор проходит по системе рекурсивных уравнений, ищет операцию останова "$\varnothing$" и пытается получить последовательность событий, которая привела к данной блокировке. Для этого анализатор запускает рекурсивный поиск всех вхождений уравнения, приводящего к ошибке, в другие уравнения системы. После получения последовательности событий, которая привела к ошибке, на основе семантических функций проводится преобразование данной последовательности в порядок команд на языке С.

Данные о верификации записываются в файл с расширением .res в виде списка, где каждая строка указывает на возможную ошибку.

Модульность структуры верификатора. Модульная система верификатора позволяет адаптировать инструмент к программам самого разного назначения, для применения в новой сфере необходимо лишь переписать front-end. Исходный код верификатора написан на языке С с использованием утилит Flex + Bison. Так как существуют реализации этих утилит для множества различных операционных систем, то данный верификатор является полностью переносимым. Однако для анализа определенных типов программ необходимо адаптировать front-end. Это связано с различной семантикой элементов синхронизации, а также особенностью реализации библиотеки для работы с потоками.

В перспективе данный подход можно обобщить на верификацию драйверов операционной системы Windows, а также использовать для анализа многопоточных и параллельных программ как для операционных систем Linux, так и для Windows. Помимо этого, в данный момент ведутся исследования, изучающие возможность применения метода из работы [2] для обнаружения состояния гонок.

Пример верификации драйвера на основе процессной семантики

Рассмотрим простейший пример драйвера устройств для операционной системы Linux, в котором имеет место взаимоблокировка. Ниже приведены наиболее

значимые части рассматриваемого драйвера, а именно операции чтения и записи:

```
static struct semaphore sem1;
static struct semaphore sem2;
static ssize_t test_read(struct file *filp, char __user *buff,
size_t count, loff_t *offp) {
    while (1) {
        down(&sem1);
        down(&sem2);
        printk ("\nIn read function");
        up(&sem2);
        up(&sem1);
}
    return count;
}
static ssize_t test_write(struct file *filp, const char __user
*buff, size_t count, loff_t *offp)
{
    while (1) {
        down(&sem2);
        down(&sem1);
        printk ("\nIn write function");
        up(&sem1);
        up(&sem2);
    }
  return count;
}
```

В данном примере используются два семафора *sem*1 и *sem*2, которые необходимы для ограничения доступа к определенному участку кода. В функции инициализации драйвера счетчикам семафоров присваивается значение 1:

```
sema_init(&sem1, 1);
sema_init(&sem2, 1);
```

Пусть данный драйвер использует программа, состоящая из двух потоков, один из которых осуществляет чтение, а другой — запись в драйвер:

```
void * writer(void *arg) {
        int fptr;
        fptr = open("/dev/test", O_RDWR | O_NONBLOCK));
        write(fptr, buf, 4);
        close(fptr);
}
void * reader(void *arg) {
        int fptr;
        fptr = open("/dev/test", O_RDWR | O_NONBLOCK));
        read(fptr, buf, 4);
        close(fptr);
}
int main() {
        pthread_t rd, wr;
        pthread_create(&wr, NULL, writer, NULL);
        pthread_create(&rd, NULL, reader, NULL);
        pthread_join(wr, NULL);
        pthread_join(rd, NULL);
        return 0;
}
```

Здесь функция *pthread_create* отвечает за создание нового потока, функция *pthread_join* ожидает завершения потока и только после этого завершает выполнение основной программы.

При запуске приложения на выполнение программа некоторое время работает, а затем происходит взаимоблокировка. Это объясняется тем, что функция чтения уменьшает счетчик семафора *sem*1 до нуля и ждет, пока функция записи увеличит значение счетчика семафора *sem*2. Но функция записи не может этого сделать, так как она ждет увеличения счетчика *sem*1.

АФС-программа для вышеприведенного примера имеет следующий вид:

```
NET
        CHAN 1 :: ALL (1) : ALL(1);
        CHAN 2 :: ALL (1) : ALL(1);
BEGIN
        FUN 1::LOOP(ALT(tt → SEQ(write(1, 1),
        write(2, 1), read(2, 1), read(1, 1))));
        FUN 2::LOOP(ALT(tt → SEQ(write(2, 1),
        write(1, 1), read(1, 1), read(2, 1))));
END.
```

Здесь *FUN* 1 соответствует операции чтения драйвера, а *FUN* 2 — операции записи. Каналы *CHAN* 1 и *CHAN* 2 моделируют семафоры *sem*1 и *sem*2 соответственно.

Определим априорную семантику функциональных процессов и каналов связи рассматриваемой АФС-программы:

$$\mathbb{F}[FUN1::LOOP(ALT(tt \to SEQ(write(1,1), write(2,1), read(2,1), read(1,1))))] =$$
$$= \mathbb{C}[LOOP(ALT(tt \to SEQ(write(1,1), write(2,1), read(2,1), read(1,1))))] =$$
$$= (\mathbb{C}[ALT(tt \to SEQ(write(1,1), write(2,1), read(2,1), read(1,1)))])^{+} =$$
$$= (\mathbb{C}[tt \to (SEQ(write(1,1), write(2,1), read(2,1), read(1,1)))])^{+} =$$
$$= (\mathbb{E}[tt] \wedge \mathbb{C}[SEQ(write(1,1), write(2,1), read(2,1), read(1,1))])^{+} =$$
$$= (T \wedge (\mathbb{C}[write(1,1)] \circ \mathbb{C}[write(2,1)] \circ \mathbb{C}[read(2,1)] \circ \mathbb{C}[read(1,1)]))^{+} =$$
$$= (T \wedge (OUT_{1,1} \circ OUT_{2,1} \circ IN_{2,1} \circ IN_{1,1}))^{+} =$$
$$= (T \wedge OUT_{1,1} \circ OUT_{2,1} \circ IN_{2,1} \circ IN_{1,1})^{+}.$$

Аналогично, для второго функционального процесса, имеем:

$$\mathbb{F}[FUN2::LOOP(ALT(tt \to SEQ(write(2,1), write(1,1), read(1,1), read(2,1))))] =$$
$$= (T \wedge OUT_{2,1} \circ OUT_{1,1} \circ IN_{1,1} \circ IN_{2,1})^{+}.$$

Построим семантическое значение каналов связи, интерпретируя их как циклический процесс:

$$\mathbb{K}[CHAN\ 1::ALL(1):ALL(1)] =$$
$$= (ALL(IN^{1,1}) \circ ALL(OUT^{1,1}))^{+} = (IN^{1,1} \circ OUT^{1,1})^{+}.$$

Аналогично для второго канала имеем:

$$\mathbb{K}[CHAN\ 2::ALL(1):ALL(1)] = (IN^{2,1} \circ OUT^{2,1})^{+}.$$

Здесь $\mathbb{F}$ и $\mathbb{K}$ — семантические функции вида $\mathbb{F}:Prog \to SP$ и $\mathbb{K}:Chan \to SP$, ставящие в соответствие функциональным процессам *Prog* и каналам связи *Chan* множества вычислительных последовательностей *SP*; *IN* и *OUT* — семантические значения команд ввода и вывода; *T* — константа, обозначающая тождественно-истинное значение; операция "$^+$" задает минимальную фиксированную точку соответствующего оператора. Отметим, что для наглядности восприятия вычислительных последовательностей используются надстрочные индексы для значений *IN*(*OUT*), характеризующих ввод (вывод) информации в канал связи, тогда как подстрочные индексы используются для значений команд ввода (вывода) информации в функциональный процесс.

Обозначим полученные априорные семантические значения через $P1$, $P2$, $K1$ и $K2$ соответственно, а семантическое значение всей программы — через *P*, тогда $P = P1 \parallel P2 \parallel K1 \parallel K2$. Система рекурсивных уравнений, характеризующая *P*, имеет следующий вид:

$$P_1 = T \wedge P_2 + T \wedge P_4;$$
$$P_2 = T \wedge P_5 + \gamma_{1,1} \circ P_8;$$
$$P_4 = T \wedge P_9 + \gamma_{2,1} \circ P_{12};$$
$$P_5 = \gamma_{1,1} \circ P_{13};$$
$$P_8 = T \wedge P_{16} + \gamma_{2,1} \circ P_{17};$$
$$P_9 = \gamma_{2,1} \circ P_{18};$$
$$P_{12} = \mathrm{T} \wedge P_{18} + \gamma_{1,1} \circ P_{22};$$
$$P_{13} = \gamma_{2,1} \circ P_{24};$$
$$P_{16} = \gamma_{2,1} \circ P_{25};$$
$$P_{17} = \mathrm{T} \wedge P_{25} + \gamma_{2,1} \circ P_{29};$$
$$P_{18} = \gamma_{1,1} \circ P_{32};$$
$$P_{22} = \mathrm{T} \wedge P_{32} + \gamma_{1,1} \circ P_{37};$$
$$P_{24} = \varnothing;$$
$$P_{25} = \gamma_{2,1} \circ P_{40};$$
$$P_{29} = T \wedge P_{40} + \gamma_{1,1} \circ P_1;$$
$$P_{32} = \gamma_{1,1} \circ P_{48};$$
$$P_{37} = T \wedge P_{48} + \gamma_{2,1} \circ P_1;$$
$$P_{40} = \gamma_{1,1} \circ P_4;$$
$$P_{48} = \gamma_{2,1} \circ P_2.$$

Из системы видно, что выполнение потоков может быть заблокировано (уравнение P_{24}). Кроме того, можно определить последовательность действий, приводящую к блокировке. В данном случае такой последовательностью является $\gamma_{1,1} \circ \gamma_{2,1}$, что следует из уравнений P_1, P_2, P_5, P_{13} и P_{24}.

Заключение

Рассмотрены вопросы, связанные с верификацией драйверов операционных систем. Описаны основные особенности существующих инструментальных средств для поиска ошибок в драйверах. Разработан инструментарий для поиска ошибок синхронизации в драйверах Linux, основанный на процессной семантике. Рассмотрены основные компоненты разработанных средств, особенности их построения.

Принципиальным отличием от инструментальных средств, приведенных в первом разделе данной статьи, является отсутствие необходимости построения модели драйвера и задания правил для проверки того или иного свойства. Это позволяет использовать данное средство без предварительной подготовки кода драйвера и без необходимости применения дополнительных программных средств.

В данный момент описанные инструментальные средства находятся на стадии активной разработки, но пока их функциональность несколько ограничена. Поэтому основная работа ведется в сторону промышленного применения данного метода. Помимо этого изучается возможность использования описанного подхода для поиска других типов ошибочных ситуаций, в частности исследуется применимость данного метода для обнаружения состояний гонок за ресурсы (*race condition*).

Список литературы

1. **Corbet J., Rubini A., Kroah-Hartman G.** Linux Device Drivers, 3-rd edition // O'Relly. 2006. 616 p.

2. **Кораблин Ю. П., Павлов Е. Г.** Разработка инструментов верификации драйверов на основе семантических моделей // Программные продукты и системы. 2012. № 1. С. 128—134.

3. **Карпов Ю. Г.** Model checking. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2009.

4. **Ben-Ari M.** Principles of the Spin Model Checker. Springer, 2008. 232 p.

5. **Mühlberg J., Lüttgen G.** BLASTING Linux Code // In the proceedings of 11th International Workshop on Formal Methods for Industrial Critical Systems. Bonn, Germany. August 26—27, 2006. P. 211—226.

6. **Clarke E., Kroening D., Lerda F.** A Tool for Checking ANSI-C Programs // In Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems. Barselona, Spain. March, 2004. P. 168—176.

7. **Ball T., Bounimova E., Kumar R., Levin V.** SLAM2: Static Driver Verification with Under 4 % False Alarms. // Proc. of Formal Methods in Computer-Aided Design. Lugano, Switzerland. October 20—23, 2010. P. 35—42.

8. **Post H., Küchlin W.** Integration of static analysis for linux device driver verification. // The 6th International Conference on Integrated Formal Methods. Oxford, UK. July 5—7. 2007. P. 518—538.

9. **Witkowski T., Blanc N., Kroening D., Weissenbacher G.** Model checking concurrent linux device drivers. // In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. Atlanta, Georgia, USA. November 5—9, 2007. P. 501—504.

10. **Clarke E., Kroening D., Sharygina N., Yorav K.** SATABS: SAT-based Predicate Abstraction for ANSI-C. // In Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems. Edinburgh, UK. April 4—8, 2005. P. 570—574.

11. **Мутилин В. С., Новиков Е. М., Страх А. В., Хорошилов А. В., Швед П. Е.** Архитектура Linux Driver Verification // Труды Института системного программирования РАН. 2011. Т. 20. С. 163—187.

12. **Кораблин Ю. П.** Семантические методы анализа распределенных систем: дис. ... д-ра техн. наук. М., 1994. 272 с.

УДК 004.4'422

В. А. Галатенко, д-р физ.-мат. наук, зав. сектором, e-mail: galat@niisi.msk.ru,
Н. И. Вьюкова, стар. науч. сотр., e-mail: niva@niisi.msk.ru,
К. А. Костюхин, канд. физ.-мат. наук, стар. науч. сотр., e-mail: kost@niisi.msk.ru,
Н. В. Шмырев, канд. физ.-мат. наук, науч. сотр., e-mail: shmyrev@niisi.msk.ru,
НИИСИ РАН, г. Москва

Опыт использования адаптивной компиляции в целях оптимизации критически важных приложений

Рассматриваются методы, направленные на адаптивное выполнение оптимизирующих преобразований с учетом данных профилирования о частоте выполнения участков кода и о значениях, принимаемых программными переменными. Рассмотрены примеры и результаты применения оптимизаций по профилю.

Ключевые слова: *оптимизация, адаптивная компиляция, профилирование*

Введение

Современные компиляторы реализуют широкий спектр оптимизаций, обеспечивающих генерацию высокоэффективного кода для различных процессорных архитектур. Однако целесообразность выполнения той или иной оптимизации для определенного фрагмента кода может зависеть не только от статических свойств программы, но и от динамических характеристик ее выполнения. Например, вынесение инвариантных вычислений из тела цикла, как правило, дает ускорение выполнения, однако, если типичное число выполнений некоторого цикла равно нулю, то данная оптимизация лишь увеличит время выполнения программы. Существуют методы оценки динамических характеристик программы на основе статического анализа ее исходного кода, однако достоверность результатов такого анализа невысока. Для решения этой проблемы в компиляторах применяются средства для получения профиля программы, который затем используется при оптимизации кода.

В контексте контролируемого выполнения авторами разработана и создана система профилирования, поддерживающая оптимизацию по профилю компилятора GCC [1] для целевых ОС семейства Linux и ОС РВ Багет, разработанной в НИИСИ РАН [2].

Средства профилирования программ изначально создавались как инструменты, позволяющие разработчику изучать профиль выполнения программы с тем, чтобы вручную оптимизировать наиболее часто выполняемые (критические) фрагменты кода. Например, чтобы в критических участках кода программист мог вручную выполнить развертку (или конвейеризацию) циклов, подстановку коротких функций вместо их вызовов и др. Современные компиляторы способны автоматически выполнять подобные оптимизации с учетом данных профилирования, оставляя разработчику содержательную деятельность по совершенствованию алгоритмических решений. Более того, с учетом профиля компиляторы выполняют множество оптимизаций, недоступных программисту на уровне исходного кода, таких как глобальное планирование команд, распределение регистров и др. По результатам различных работ (см., например, [3, 4]) повышение эффективности кода в результате применения оптимизаций по профилю может составлять 10...20 % и более.

Виды профилей и их использование при оптимизации кода

Существуют различные виды профилей, которые могут быть полезны для тех или иных оптимизаций. Профиль потока управления (*control flow profile*) может быть представлен с различной степенью детализации:

профиль функций (*function profile*) содержит информацию о частоте выполнения отдельных функций; профиль базовых блоков (*basic block profile*) — об ожидаемом числе выполнений каждого блока; профиль переходов (*edge profile*) — о вероятностях переходов (дуг графа управления); профиль путей (*path profile*) — о частоте выполнения отдельных конечных путей в графе управления.

Наиболее информативным является профиль путей, однако его сбор требует более высоких накладных расходов, а представление этого профиля сложнее, чем для других видов. По профилю переходов можно вычислить относительные частоты базовых блоков, но нельзя в общем случае получить профиль путей, поскольку вероятность перехода представлена в нем вне зависимости от предыстории выполнения. Тем не менее, именно профиль переходов наиболее часто используется в компиляторах в качестве разумного компромисса между ценой и качеством. Как показано в работе [3], профиль переходов более информативен, чем профиль блоков, притом, что связанные с его сбором накладные расходы ненамного выше, чем для профиля блоков.

Другой вид профиля — профиль значений — содержит информацию о значениях, принимаемых некоторыми программными выражениями. Этот вид профиля представляет собой гистограмму значений, принимаемых выражением.

Сбор профиля может быть реализован как аппаратными средствами, так и программно. Преимущества аппаратной реализации — низкие накладные расходы и возможность получать профили приложений прозрачным для разработчика образом. Однако точность получаемого профиля может быть ниже и могут быть доступны не все виды профилей. Преимущества программной реализации профилирования — гибкость и доступность требуемых видов профиля независимо от аппаратных возможностей процессора.

Процедура для применения оптимизаций по профилю, собираемому программным способом, включает три шага:

- инструментирование программы (компиляция программы выполняется с ключами, задающими сбор профиля);
- сбор профиля (инструментированная программа выполняется с некоторыми типичными наборами входных данных, при этом накапливается и сохраняется ее профиль);
- компиляция с использованием профиля (компиляция выполняется с ключами, задающими использование профиля при оптимизациях).

Помимо специальных оптимизаций, ориентированных на применение профиля, компилятор может использовать профиль для повышения эффективности многих традиционных оптимизаций [5].

Частоты выполнения базовых блоков и профиль переходов могут применяться для следующих целей:

- принятие решений о целесообразности применения таких оптимизаций как inline-подстановка функций, развертка или пилинг цикла;
- выявление циклов, не оформленных в виде соответствующих языковых конструкций;
- трансформации циклов;
- распределение регистров;
- повышение локальности кода для эффективного использования кэша команд (переупорядочение функций и участков кода в соответствии с типичными последовательностями их выполнения);
- размещение функций (и даже отдельных фрагментов функций) в разных секциях в зависимости от частоты их выполнения (.text, .text.hot, .text.unlikely_executed).

Профиль значений может быть использован следующими оптимизациями:

- специализация операций;
- частичное применение функций;
- предвыборка данных;
- реорганизация циклов [6];
- повышение локальности данных для эффективного использования кэша (реорганизация структур, матриц);
- реализация операторов-переключателей (*switch*): если значения выражения, по которому происходит переключение, распределены более или менее равномерно, то наиболее эффективной реализацией оператора является таблица переходов; если некоторые значения встречаются значительно чаще других, то предпочтительный способ реализации — последовательность условных переходов, упорядоченных по частоте соответствующих значений.

Важный частный случай профиля значений — профиль зависимостей по памяти (см., например, [7]). При планировании потока команд (особенно при глобальном планировании и при конвейеризации циклов) зависимости по памяти препятствуют переносу команд загрузки выше предшествующих им команд записи. Поскольку команды загрузки часто оказываются на критическом пути, подобные зависимости ограничивают возможности генерации эффективного кода. Статический анализ программ часто не позволяет определить, что зависимость по памяти в данном случае фактически отсутствует, поэтому компилятор вынужден считаться с возможностью совпадения адресов в командах записи и чтения. Возможно также, что совпадение адресов при выполнении программы возникает лишь в редких случаях. Если профиль зависимостей показывает, что совпадение адресов для некоторой пары команд записи-чтения случается редко (или никогда), то компилятор может переместить чтение выше записи, добавив код для проверки адресов и выполнения компенсационных действий в случае совпадения адресов (который будет выполняться лишь в редких случаях). Наиболее эффективен данный подход при наличии аппаратной поддержки необходимых проверок в виде буфера конфликтов памяти (*memory conflict buffer*, MCB), однако возможна и программная реализация [8].

Оптимизации по профилю в компиляторе GCC

В компиляторе GCC программный сбор профиля и оптимизации по профилю поддерживаются, начиная с версии 4.0. Используются профиль вероятностей переходов и профиль значений. В GCC реализована инструментовка выполняемой программы для сбора статистических характеристик, а именно:

- счетчик числа выполнений базового блока;
- счетчик числа выполнений перехода;
- детектор степеней двойки (позволяет определить, что некоторое выражение принимает значения, равные степеням 2);
- детектор единственного значения (позволяет определить, что некоторое выражение почти всегда принимает одно и то же значение);
- детектор постоянной разницы (позволяет определить, что интервал между значениями, последовательно принимаемыми некоторым выражением, практически всегда постоянен);
- детектор постоянного адреса вызова (позволяет определить, что адрес косвенного вызова функции с большой вероятностью принимает одно и то же значение);
- счетчик среднего значения (вычисляет среднее значение выражения);
- детектор выравнивания (позволяет определить ожидаемое выравнивание адресного выражения).

Во внутреннем представлении программы профиль потока управления хранится в виде полей структур, описывающих узлы и дуги графа потока управления (CFG — *Control Flow Graph*). Для базовых блоков поддерживаются поля счетчика и относительной частоты выполнения; для переходов — поля счетчика и вероятности перехода. При этом для базовых блоков поддерживаются предикаты maybe_hot_p (часто выполняемый), probably_cold_p (редко выполняемый) и probably_never_executed (с большой вероятностью не будет выполнен ни разу). Проходы оптимизации в GCC были изменены таким образом, чтобы данные профилирования при всех преобразованиях CFG поддерживались в актуальном состоянии. Эти предикаты опрашиваются различными оптимизациями, которым важна информация об относительной частоте выполнения базовых блоков, поскольку они обеспечивают

Кросс-компиляция с оптимизацией по профилю, полученному на инструментальной платформе

повышение эффективности кода за счет увеличения его размера. В компиляторе GCC в настоящее время профиль потока управления используется при принятии следующих решений:

- о развертке и пилинге циклов;
- об inline-подстановках функций;
- при распределении регистров для определения приоритетов выделения физических регистров виртуальным.

Профиль значений хранится в виде аннотаций к элементам внутреннего представления программы. На основе профиля значений компилятор выполняет следующие оптимизации:

- специализация операций деления и взятия по модулю: если можно определить, что операнды имеют некоторые специальные значения, то можно реализовать эти дорогостоящие операции более эффективно;
- спекулятивная предвыборка данных: если можно определить, что интервалы между значениями адресных выражений, по которым осуществляется обращение к памяти, постоянны, то в код добавить инструкции предвыборки данных;
- специализация косвенных вызовов функций: если адрес вызова постоянен, то можно выполнить inline-подстановку вызываемой функции;
- специализация стандартных функций: если операнды таких функций как memcpy, strcpy выровнены по границе слова, то для этих функций можно сформировать более эффективный код.

Для всех оптимизаций по профилю значений компилятор формирует альтернативные ветви выполнения, а именно добавляется проверка условия, при котором данная оптимизация применима, и код генерируется как для частного, так и для общего случаев.

Для генерации инструментированного кода в целях получения профиля необходимо скомпилировать программу с ключом-fprofile-generate. Оптимизация кода по профилю осуществляется при задании ключа, важна информация об относительной частоте выполнения базовых блоков (если задан ключ-fprofile-use, а файлы с профилем отсутствуют, то компилятор будет использовать статические оценки частоты переходов). При этом возможно получение профиля на одной платформе и его использование при кросс-компиляции той же программы для другой платформы — важно лишь, чтобы версии применяемых компиляторов совпадали. Авторы проводили эксперименты с компилятором GCC[1] для процессоров i686 под управлением ОС Linux (инструментальная платформа) и кросс-компилятором GCC, выполняющемся на инструментальной платформе и генерирующем код для процессора mips под управлением ОС реального времени, см. рисунок.

В таблице приведены результаты для тестов FLOPS и LINPACK, полученных путем кросс-компиляции с оптимизацией по профилю, вычисленному на целевой платформе i686/Linux.

[1] Использовались версии GCC 3.4.5 и экспериментальный вариант GCC 4.2.

Результаты тестирования компилятора GCC, флопс

Набор тестов	Результат без оптимизации по профилю	Результат с оптимизацией по профилю
FLOPS		
(1)	80,3089	109,1684
(2)	68,1590	85,5698
(3)	95,1043	131,2913
(4)	117,7874	191,0134
LINPACK	53,230	57,703

Рассмотрим несколько примеров оптимизаций по профилю, поддерживаемых в GCC.

Адаптивные inline-подстановки. Компилятор проводит подстановку функций в часто выполняемых участках кода, но не делает ее в редко выполняемых участках. Это позволяет значительно сократить размер кода без потери производительности. Пример:

```
int b[100];
void abort (void);
inline void
cold_function ()
{
  int i;
  for (i = 0; i < 99; i++)
    if (b[i] / (b[i + 1] + 1))
      abort ();
}

inline void
hot_function ()
{
  int i;
  for (i = 0; i < 99; i++)
    if (b[i] / (b[i + 1] + 1))
      abort ();
}

main ()
{
  int a;

  for (a = 0; a < 1000; a++) {
    if (a == 0)
        cold_function ();
      else
        hot_function ();
  }
  return 0;
}
```

В этом примере будет выполнена подстановка тела функции `hot_function`, а функция `cold_function` не будет подставлена.

Подстановка косвенных вызовов. Если указатель, по которому проводится косвенный вызов функции, с большой вероятностью принимает некоторое постоянное значение *f*, и соответствующая функция доступна компилятору для inline-подстановки, то для такого косвенного вызова будет сформирована условная ветвь с подстановкой часто вызываемой функции. Пример:

```
static int a1 (void)
{
  return 10;
}

static int a2 (void)
{
  return 0;
}
typedef int (*tp) (void);

static tp aa [] = {a2, a1, a1, a1, a1};

void setp (int (**pp) (void), int i)
{
  if (!i)
    *pp = aa [i];
  else
    *pp = aa [(i & 2) + 1];
}

int
main (void)
{
  int (*p) (void);
  int i;

  for (i = 0; i < 10; i ++)
    {
      setp (&p, i);
      p ();
    }

  return 0;
}
```

Здесь для вызова `p ()` в функции `main` будет сформирован код вида

```
if (p = = a1)
  <inline-подстановка функции a1>
else
  p ();
```

Специализация операций. Дорогостоящие операции, такие как целочисленное деление или взятие остатка, при некоторых значениях операндов могут быть реализованы более эффективно. Для таких операций компилятор (при наличии ключа-fprofile-generate) обеспечивает сбор профиля. Если операнды с большой вероятностью принимают "хорошие" значения, то компилятор (при наличии ключа-fprofile-use) сгенерирует ветви для быстрого вычисления результата. Пример:

```
int a[1000];
int b = 256;
int c = 257;
main ()
{
  int i;
  int n;
  for (i = 0; i < 1000; i++)
    {
      if (i % 17)
        n = c;
      else n = b;
      a[i] / = n;
    }
  return 0;
}
```

Здесь в операторе `a[i] /= n` делитель будет преимущественно принимать значение 257, поэтому компилятор сгенерирует ветвь для быстрого вычисления результата (путем умножения делимого на некоторую большую константу).

На процессоре i686 использование оптимизации по профилю сокращает время выполнения данного теста примерно на 25 % (0,29 с против 0,39 с).

Специализация встраиваемых стандартных функций. Компилятор может подставлять код для стандартных функций языка С; он может оптимизировать этот код, если аргументы функции в данном вызове принимают специальные значения. Например, если профиль значений показывает, что аргументы таких функций как `memset, memcopy` кратны 4 (8) байтам, то вызовы этих функций могут быть реализованы как последовательности пересылок слов (двойных слов), например:

```
memcopy (a, b, 4) -> a[0] = b [0]
memset (a, 10, 4) -> a[0] = ...
```

В целом, по результатам работы [4], эффект от оптимизаций по профилю в GCC для целевой платформы i386 составил 12 % на тестах SPECint. В настоящее время в GCC реализована полноценная инфраструктура для поддержки оптимизаций по профилю, продолжается работа по совершенствованию имеющихся оптимизаций с учетом профиля, открыты проекты по реализации новых оптимизаций по профилю [5].

Другие подходы к адаптивной компиляции

Динамическая адаптивная реоптимизация. Хотя оптимизация по профилю может давать значительный прирост производительности и способствовать сокращению размера кода, с ее применением связан ряд проблем. Необходимость двукратной компиляции, а также разработки и сопровождения достаточно представительных тестовых задач для сбора профиля усложняют процесс разработки и сопровождения программ. Не всегда возможно предугадать, с какими входными данными приложение будет работать в производственных условиях (возможно, разные группы пользователей будут использовать приложение по-разному). Кроме того, как показывает практика, в некоторых случаях оптимизация по профилю может приводить к деградации производительности каких-то частей программы, т. е. разработчику необходимо анализировать результаты оптимизации по профилю и, возможно, применять ее избирательно.

Для решения этих проблем применяется метод динамической адаптивной реоптимизации программ. Реализации этого подхода представлены в работах [9—11]. Рассмотрим в качестве примера инфраструктуру динамической реоптимизации, описанную в работе [9]. Когда пользователь впервые запускает приложение, компонент загрузки и генерации кода транслирует представление этой программы в машинный код. Для того чтобы ускорить процесс запуска, на этой стадии оптимизация приложения не выполняется. Компонент профилирования обеспечивает сбор данных о динамике выполнения приложения. Он собирает различные частотные характеристики, а также данные о наиболее часто используемых значениях параметров при вызовах функций. Компонент управления средой реоптимизации (выполняющийся в рамках низкоприоритетного потока) периодически опрашивает базу данных профилирования на предмет изменений профилей отдельных приложений и формирует очередь приложений для реоптимизации. Приложения в очереди упорядочиваются в соответствии с ожидаемым эффектом от их реоптимизации и передаются на вход компонента оптимизации. По завершении оптимизации очередного приложения, загруженный код этого приложения замещается оптимизированным кодом. Подобная схема допускает использование разных методов профилирования и представлений программ, содержащих достаточно информации для ее оптимизации.

Данный подход избавляет разработчиков приложений от необходимости создавать и поддерживать тестовые задачи и не требует сложной схемы сборки приложений. Он также обеспечивает адаптацию кода программы к реальной динамике выполнения программы. Вместе с тем использование динамической реоптимизации требует наличия специальной довольно сложной среды выполнения. К тому же она может противоречить требованиям надежности, поскольку код приложения изменяется "на лету" непосредственно во время выполнения. По этим причинам данный подход неприменим, в частности, для встроенных систем.

Адаптивный подбор последовательностей оптимизирующих преобразований. Суть этого подхода заключается в индивидуальном подборе последовательностей оптимизирующих преобразований для отдельных участков кода. Действительно, не существует универсальной последовательности, которая обеспечивала бы наилучший результат для любого участка кода. Например, один цикл выгоднее развернуть, другой — конвейеризовать, для третьего не имеет смысла ни то, ни другое; в одних случаях предвыборка данных ускоряет выполнение цикла, в других — замедляет; обычно бывает выгодно выполнить планирование кода до и после распределения регистров, но иногда первое планирование лучше отменить, поскольку оно может приводить к нехватке физических регистров и т. п.

К приложениям для встроенных систем зачастую предъявляются жесткие требования по производительности и/или энергопотреблению. Обычно также существуют ограничения на размер кода. В таких условиях оправдано применение достаточно сложных и трудоемких средств компиляции, позволяющих приблизить генерируемый код по указанным характеристикам к программам, написанным на языке ассемблера квалифицированным программистом. К числу таких средств относится метод адаптивного подбора оптимизирующих последовательностей, состоящий в том, чтобы при помощи какой-либо стратегии перебора поставить в соответствие каждому участку программы некоторую последовательность оптимизаций

(из фиксированного набора) так, чтобы получить "достаточно хороший" по заданным характеристикам код. Поскольку пространство перебора слишком велико, обычно не ставится задача полного перебора для нахождения точного оптимума, как правило, применяют алгоритмы генетического перебора, жадные алгоритмы или перебор "по градиенту" и ограничивают время поиска решения. В качестве участков адаптивной компиляции могут выступать базовые блоки, функции, циклы или гнезда циклов. Как правило, среда адаптивной компиляции предоставляет интерактивную оболочку, позволяющую пользователю управлять процессом компиляции (например, вручную выбирать набор оптимизаций для отдельных участков, задавать метод и время перебора и т. п.).

Примеры реализаций этого подхода представлены в работах [12—14]. Реализации различаются наборами адаптивно применяемых оптимизирующих преобразований и некоторыми другими характеристиками. Так в работе [12] участками адаптивной компиляции являются функции, а набор оптимизаций включает исключение мертвого кода, исключение общих подвыражений, подбор команд, распределение регистров и другие оптимизации, ориентированные на специфику целевого процессора. Для оценки эффективности сгенерированного кода программа выполняется на целевом процессоре с замером числа выполненных инструкций. В работе [13] в адаптивной компиляции используется более широкий набор оптимизаций, включающий все виды оптимизаций, реализованных в базовом компиляторе, в том числе оптимизации циклов.

Хотя адаптивная компиляция "в чистом виде" не опирается на данные профилирования, их использование может быть весьма полезно. Например, в работе [13] профиль частоты выполнения базовых блоков используется для того, чтобы сэкономить время, затрачиваемое на перебор оценочных прогонов вариантов генерируемого кода. Программа выполняется всего один раз с получением профиля, а эффективность ее вариантов, полученных с применением различных оптимизирующих последовательностей, оценивается на основе данных о частоте выполнения базовых блоков.

Профиль можно было бы также учитывать при выборе множества оптимизаций для различных участков программы, что фактически означает интеграцию обычной оптимизации по профилю с переборной адаптивной компиляцией; разумно было бы также варьировать интенсивность перебора оптимизирующих последовательностей для различных участков программы с учетом частоты их выполнения.

Заключение

Данные профилирования являются важным источником информации для компилятора при принятии решений о целесообразности и параметрах различных оптимизаций. Повышение эффективности кода в результате применения оптимизаций по профилю может составлять в среднем 10...20 %, а для отдельных программ эффект может быть значительно выше. В настоящее время ведутся активные исследования в области оптимизации по профилю, в частности по такому направлению как динамическая реоптимизация приложений по профилю. В сфере компиляции для встроенных систем развиваются переборные методы адаптивной компиляции, позволяющие подбирать оптимизирующие последовательности индивидуально для различных участков программы. В связи с этим обстоятельством интересным направлением исследований представляется интеграция традиционной оптимизации по профилю с переборными методами адаптивной компиляции.

Список литературы

1. **GCC:** the GNU Compiler Collection. URL: http://GCC.gnu.org.

2. **Вьюкова Н. И., Галатенко В. А., Костюхин К. А., Шмырев Н. В.** Организация отладочного комплекса для целевых систем со сложной архитектурой. М.: Изд-во НИИСИ РАН, 2004.

3. **Cox J. S., Howell D. P., Conte T. M**. Commercializing Profile-Driven Optimization // Proceedings of the 28th Annual Hawaii Intemotional Conference on System Sciences. 1995. P. 221—228.

4. **Hubicka J.** Profile driven optimizations in GCC // In GCC Developers Summit Procedings. Ottawa, Ontario, Canada. June 2005. P. 107—124.

5. **Chang P. P., Mahlke S. A., and Hwu W. W.** Using profile information to assist classic code optimizations // Software: Practice and Experience. 1991. N 21 (12). P. 1301—1321.

6. **Gunter R.** Profile driven loop transformations // In GCC Developers Summit Procedings. Ottawa, Ontario, Canada. June 2006. URL: http://ols.fedoraproject.org/GCC/Reprints-2006/guenther-reprint.pdf

7. **Chen W. Y., Mahlke S. A., Warter N. J., Anik S., Hwu W. W**. Profile-Assisted Instruction Scheduling // International Journal of Parallel Programming. 1994. Vol. 22. N 2. P. 151—181.

8. **Nicolau A.** Run-time disambiguation: coping with statically unpredictable dependencies // IEEE Transactions on Computers. 1989. Vol. 38. P. 663—678.

9. **Kistler T., Franz M.** Continuous Program Optimization: A Case Study // ACM Transactions on Programming Languages and Systems. 2003. Vol. 25. N 4. P. 500—548.

10. **Lattner C. A.** LLVM: An Infrastructure for Multi-Stage Optimization. Thesis. University of Illinois at Urbana-Champaign, 2002.

11. **Burger R. G., Dybvig R. K.** An Infrastructure for Profile-Driven Dynamic Recompilation // Proceedings of ICCL'98. Chicago, Illinois. May 1998. P. 240—251.

12. **Kulkarni P., Zhao W., Moon H.** et al. Finding Effective Optimization Phase Sequences // LCTES'03. San Diego, California, USA. June 11—13, 2003. P. 12—23.

13. **Cooper K. D., Grosul A., Harvey T.J., Reeves S., Subramanian D., Torczon L., and Waterman T.** ACME: Adaptive Compilation Made Efficient // LCTES'05. Chicago, Illinois, USA. June 15—17, 2005. P. 69—77.

14. **Hines S., Kulkarni P., Davidson J., Whalley D.** Using De-optimization to Re-optimize Code // EMSOFT'05. Jersey City, New Jersey, USA. September 19—22, 2005. P. 114—123.

УДК 004.432

А. Г. Бельтов[1], канд. техн. наук, доц., ген. директор,
И. Ю. Жуков[1], д-р техн. наук, проф., зам. ген. директора, e-mail: i_zhukov@cniiesu.ru,
Д. М. Михайлов[2], канд. техн. наук, доц., e-mail: mr.mdmitry@gmail.com,
А. В. Зуйков[2], аспирант, e-mail: avzuykov@gmail.com,
М. И. Фроимсон[2], аспирант, e-mail: froimsonm@gmail.com,
А. М. Рапетов[2], студент, e-mail: willir29@yandex.ru,
А. А. Кузин[2], студент, e-mail: akudiyar@gmail.com,
[1]ФГУП "ЦНИИ ЭИСУ", г. Москва,
[2]НИЯУ МИФИ, г. Москва

Эффективность использования языков программирования C++ и Java для разработки программного обеспечения для ОС Android

Данная статья посвящена анализу эффективности написания приложений, работающих под управлением операционной системы (ОС) Android, на языках программирования Java и C++. Рассматриваются такие аспекты, характеризующие эффективность программирования, как механизмы использования памяти и скорость выполнения программы. Анализируются преимущества и недостатки языков Java и C++, после чего дается заключение о целесообразности смешанного использования языков программирования.

Ключевые слова: *Java, C/C++, производительность, декомпиляция, эффективность, память, код*

Данная статья посвящена исследованию эффективности использования языков программирования C++ и Java для разработки программного обеспечения для ОС Android. Выбор этих языков обусловлен тем, что разработка полноценных приложений под ОС Android возможна только на языках C++ и Java. Причина в том, что компания Google предоставляет стандартный комплекс средств разработки именно для языков Java и C/C++. При этом язык Java был выбран компанией Google как основной язык для написания высокоуровневых библиотек и приложений под ОС Android, а использование языка C/C++ предполагается как второстепенное, для написания критичных к производительности участков кода.

Выбор ОС Android для оценки производительности языков при написании программ для этой ОС обусловлен ее популярностью, а также тем обстоятельством, что она является открытой и позволяет решать широкий круг задач. В том числе она позволяет скомпилировать Android из исходных данных, доступных в открытом виде.

Не секрет, что ОС Android "заточена" под язык программирования Java. Возможность написания приложений на языке C++ под Android очень ограничена — можно выполнять только небольшой набор операций. Это связано с тем что, взаимодействие на уровне библиотек и приложений ОС Android выполняется в среде виртуальной Java-машины Dalvik. Чтобы использовать мультимедиаресурсы телефона, например, камеру, необходимо обращаться к Java-коду, так как для языка C++ не написано соответствующих библиотек. На языке C++ можно реализовывать логику, работу с файлами, математические расчеты, т. е. то, что не связано с интерфейсом и ресурсами телефона, потому что для встраиваемого C++ кода доступ к ним запрещен. Последнее обстоятельство создает большие трудности написания приложений и значительные накладные расходы.

Несмотря на отмеченные выше трудности разработки приложений под ОС Android, существуют практически значимые прикладные области, в которых без языка C++ не обойтись.

Цель данной статьи — сравнить преимущества написания программ на языках Java и C++ для ОС Android. Рассматриваются сильные и слабые стороны языков C++ и Java, их производительность в процессе написания приложений для операционной системы Android.

Используются следующие критерии, по которым проводится сравнение:

- сложность декомпиляции программы злоумышленником в целях исследования алгоритма;
- количество системных ресурсов, которые необходимы приложению, разработанному на выбранном языке;
- простота реализации задачи на выбранном языке;
- полнота поддержки функционала периферии устройства;
- простота отладки приложения.

Защита от декомпиляции

Известно, что процесс декомпиляции (англ. *reverse-engineering* — обратная разработка) Java-кода из .class файлов, в виде которых существуют программы на Java, не сложный. Этот факт делает его доступным для злоумышленников, которые могут узнать об "узких" местах в вашей программе или, например, изучить закрытые алгоритмы. Для защиты от декомпиляции часто применяется обфускация.

Обфускация представляет собой приведение исходного текста или исполняемого кода программы к виду, сохраняющему ее функциональность, однако затрудняющему анализ, понимание алгоритмов работы приложения и модификацию кода при декомпиляции.

Обфускация, или "запутывание" кода может осуществляться на уровне алгоритма, исходного текста или ассемблерного текста.

В этот процесс входит переименование и запутывание логики — замена блоков на более простые конструкции и перемещение вызовов [1, 2].

Методы изменения логики называются общим термином *flow-obfuscation*. Обфускатор без реализации этих методов не гарантирует защиту кода от декомпиляции, так как даже после переименования всех классов и всех строк можно понять алгоритм и назначение кода. Существующие обфускаторы с *flow-obfuscation*, например, Zelix, Allatori, являются коммерческими. В связи с этим их использование значительно увеличивает затраты на изготовление программного продукта.

Зачастую в обфускаторы, несмотря на тщательное проектирование и тестирование, вкрадываются ошибки, что делает их работу нестабильной. Как следствие, существует вероятность того, что прошедший через обфускатор Java-код вообще не будет работать. И чем сложнее разрабатываемая программа, тем больше эта вероятность. Код после обфускации может стать более зависимым от платформы или компилятора [2]. Таким образом, обфускация не дает полной гарантии того, что код нельзя будет декомпилировать.

Большинство языков с промежуточным кодом могут создавать или вызывать объекты по именам их классов. Современные обфускаторы позволяют сохранить указанные классы от переименования, однако подобные ограничения сокращают гибкость программ.

В таких широко используемых компилируемых языках как C/C++, не "заточенных" под переносимость, исполняемый код машинозависим и очень сложен. Его нельзя разобрать математически.

Решение проблемы сокрытия кода может быть следующее — использование сочетания языков C++ и Java. Критичные участки кода, которые необходимо скрыть, можно написать на C++, скомпилировать и использовать в виде библиотек (в формате .so, типичном для UNIX). В результате использования данного подхода получается машинозависимый код, который крайне тяжело декомпилировать. При этом скомпилированные библиотеки C++ подключаются в Java с помощью интерфейса JNI (*Java Native Interface*).

Производительность

В обычных компилируемых языках исходный код сразу преобразуется в код, понятный конкретному процессору. Компилятор Java преобразует свой код из исходных данных в байт-код, который исполняется на специальной виртуальной машине (*Java Virtual Machine*, JVM), — программе, обрабатывающей байт-код и передающей инструкции оборудованию как интерпретатор. JVM является промежуточным пунктом между исходным кодом и процессором [3].

Достоинство подобного способа выполнения программ заключается в полной независимости байт-кода от ОС и оборудования. Это обстоятельство позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина.

Язык Java обладает следующими возможностями, недоступными для C++, либо доступными с серьезными ограничениями [4]:

- автоматическое управление памятью;
- богатый набор средств фильтрации ввода/вывода;
- наличие простых средств создания сетевых соединений;
- расширенные возможности обработки исключительных ситуаций;
- наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы;
- встроенные в язык средства создания многопоточных приложений;
- встроенный в библиотеку Android унифицированный доступ к базам данных, и рядом других возможностей.

Другой важной особенностью технологии Java является гибкая система безопасности. Такая гибкость обеспечивается благодаря тому, что исполнение программы полностью контролируется виртуальной машиной.

В то же время подобная "прослойка" может серьезно повлиять на производительность программы.

Проанализируем производительность программ, написанных на языках программирования C++ и Java с точки зрения следующих двух значимых параметров, определяющих эффективность работы системы:

- подход к использованию памяти;
- скорость выполнения программ.

Подход к использованию памяти. Первым фактором, влияющим на производительность системы, является объем затрачиваемой на выполнение приложения памяти, так как у многих мобильных телефонов есть строгое ограничение по этому параметру. Если его превысить, приложение просто не будет работать. Языки программирования Java и C++ используют различные подходы в управлении памятью.

В языке C++ управление памятью полностью осуществляется программистом, который при необходимости перераспределяет или освобождает память. Если этого не происходит своевременно, возникает "утечка памяти". Если во время работы приложения произойдет лишь одна такая утечка, после завершения работы приложения ОС освободит всю ранее использованную им память и система продолжит стабильную работу. Однако если "утечки памяти" будут происходить постоянно (например, если пользователь будет периодически выполнять определенные действия), то использование памяти приложением будет расти вплоть до полного ее израсходования с последующим возможным отказом системы. С программами, написанными на языке C++, часто возникают ситуации, когда разработчик забывает освободить память, информация накапливается, и эту ошибку очень тяжело отслеживать и исправлять.

В свою очередь, в языке Java обеспечивается автоматическое освобождение неиспользуемой памяти [3]. Это делается с помощью специального "сборщика мусора". Однако у "сборщика мусора" есть и отрицательные стороны.

Наряду с распределением памяти программистом, JVM ведет учет всех используемых блоков памяти. Если какой-то блок памяти больше не используется, он может быть освобожден. На каждом такте вызова "сборщик мусора" проходит всю память и помечает объекты. Те объекты, что были давно отмечены, удаляются.

"Сборка мусора" очень удобна, однако этот процесс потребляет большое количество памяти, что приводит к снижению производительности.

Программисты, работающие с языком C++, могут освобождать блоки памяти сразу после того, как в их использовании пропала необходимость. При работе с языком Java блоки не освобождаются до очередного вызова "сборщика мусора", периодичность работы которого зависит от используемой реализации JVM.

Кроме большого расхода памяти, процесс "сборки мусора" требует дополнительной процессорной мощности. В результате, когда такой ресурс становится недоступен приложению, это приводит к замедлению его работы. В силу отмеченных факторов периодическая работа "сборщика мусора" может стать причиной "замораживания" Java-программы.

Также стоит отметить, что структуры/классы на языке Java занимают больше памяти, чем подобные структуры/классы на языке C++. Это связано с тем, что на языке Java все методы по умолчанию являются виртуальными, следовательно, во всех классах языка Java присутствуют дополнительные данные (заголовок), необходимые для правильного вызова виртуальных методов от базовых классов и динамического привидения типов. Размер заголовка зависит от разрядности. В мобильных устройствах используется 32-разрядный ARM-процессор, и, следовательно, размер заголовка в этом случае равен 8 байтам [5, 6].

В языке C++, если в классе/структуре отсутствуют виртуальные методы, то никаких дополнительных данных не присутствует. И даже если виртуальные методы наличиствуют, то в класс неявно добавляется один указатель на таблицу виртуальных функций, размер которого зависит от разрядности ОС. В мобильных устройствах используется 32-разрядный ARM-процессор, и в этом случае размер указателя равен 4 байтам.

Тот факт, что Java-программы используют больше памяти, чем аналогичный код на языке C++, является особенно критичным для встраиваемых и мобильных устройств с небольшим объемом памяти. В среднем для смартфона HTC Legend объем памяти, выделяемой каждому приложению, — 16...32 Мбайт. Зачастую складывается ситуация, когда этого не хватает для решения конкретных практических задач.

С учетом изложенных выше соображений, отрицательные моменты использования языка Java состоят в неэффективности механизмов работы с памятью. Однако, и это следует отметить, они снимают с разработчика обязанность следить за состоянием памяти.

Скорость выполнения программ. Другим фактором, характеризующим эффективность работы ОС, является производительность. Скомпилированный C++ код хранится в двоичном формате и поэтому выполняется непосредственно процессором, т. е. аппаратными средствами.

Компилятор Java преобразует объектный код в байт-код, который исполняется с помощью отдельного программного обеспечения — виртуальной машины Java. В свою очередь, эта машина (JVM) сама исполняется процессором. Выполнение байт-кода Java-программ осуществляется не быстрыми аппаратными средствами, а с помощью более медленной программной эмуляции, что значительно увеличивает время отклика приложения. Так как все функции Java проходят через Java-машину, время их вызова увеличивается в среднем в 2—3 раза, что сказывается на скорости работы приложения [7]. На машинах ограниченной производительности время задержки увеличивается.

Для подтверждения сделанных заключений были проведены тестирования для таких базовых операций как вызов метода и создание объекта "в куче". В качестве тестирования времени вызова метода был выбран алгоритм рекурсивного поиска максимального числа в массиве случайных чисел разрядностью 32 бита. На устройстве Samsung Galaxy R были получены данные, представленные в табл. 1—2.

Таблица 1

Результаты тестирования скорости создания объекта "в куче"

Число повторений	Время выполнения, мс	
	Java	C++
1 000 000	1900 ± 130	500 ± 20
5 000 000	10 000 ± 400	2500 ± 70

Таблица 2

Результаты тестирования скорости алгоритма рекурсивного поиска максимального числа по массиву чисел разрядностью 32 бита

Размер массива	Время выполнения, мс	
	Java	C++
1 МБайт	360 ± 12	40 ± 2
5 Мбайт	1560 ± 50	210 ± 5

Как видно из представленных таблиц, производительность приложений, написанных на языке Java, сильно отстает от производительности приложений, написанных на языке C++, для таких базовых операций, как вызов метода и создание объекта. Так как эти операции являются основой любой программы, то общая производительность программ, написанных на языке Java, будет ниже, чем у аналогичных программ, написанных на языке C++.

Простота реализации задачи на выбранном языке

Написание программ на языке C++ под ОС Android связано с определенными сложностями, в числе которых описанные ниже.

- Недостаток в литературных источниках, так как значительная часть учебных пособий и руководств по написанию программного обеспечения под ОС Android посвящена языку Java.
- В подавляющем большинстве случаев вызов C++ кода необходимо совершать с помощью Java через механизм JNI. Исключением является возможность написания только ограниченного интерфейса приложения на C++ (без использования Java).
- Компиляция C++ исходных кодов требует дополнительного программного обеспечения (NDK, *Native Development Kit*) и выполняется сложнее, чем компиляция исходных Java-кодов.
- Отсутствие хорошей среды разработки для написания программ на языке C++ под ОС Android.

Следует также отметить, что язык Java обладает большей степенью абстракции и лучше структурирован, чем C++. В связи с этим написание программ на языке Java проще, реализуется быстрее, чем на C++, и требует меньшего объема кода. Язык Java прост в изучении, и написание стабильных программ на Java требует меньших уровня знаний и навыков, так как Java менее склонен к ошибкам.

Возможность работы с периферией устройства с использованием языков Java и C++

Подавляющее большинство библиотек написано на языке Java. Существует возможность вызова библиотек через механизм рефлексий из C++ кода. Однако его реализация сопряжена с большими сложностями и потерей производительности (время вызова метода увеличивается до 20 раз). В частности, Android-библиотеки позволяют получать доступ к периферии устройства. Следовательно, класс задач, реализуемых на языке C++, очень ограничен.

Простота отладки приложения

Отладка приложений для ОС Android, написанных на языке Java, более проста в реализации и удобна, так как лучше документирована. При этом отладка не требуют дополнительных знаний.

Необрабатываемые ошибки в работе программы вызывают исключительные ситуации, при этом:

- лог исключительной ситуации, произошедшей в программе, написанной на языке Java, содержит полный Stack Trace с именами методов и номеров строк, в которых произошла исключительная ситуация, а также текстовое описание причины исключительной ситуации;
- лог исключительной ситуации, произошедшей в программе, написанной на языке C++, содержит только адреса команд ассемблера, в которых произошла исключительная ситуация. Утилита (*ndk-stack*), преобразующая адреса в читаемый вид, была добавлена только в 2011 г., поэтому она не всегда корректно работает.

Для обоих языков существует возможность пошаговой отладки, однако управление ею для языка C++ реализовано через консоль, поэтому и работа с ней менее удобна, чем с аналогичной утилитой для языка Java.

Выводы

В результате проведенного анализа были выявлены сильные стороны языков Java и C++ для написания приложений под платформу Android.

Для языка Java характерны следующие преимущества:

- присутствует множество библиотек для работы с *User Interface* (UI), а также периферией мобильного устройства;
- программирование на языке Java несколько проще, чем на C++;
- отладка кода, написанного на языке Java, значительно проще.

Преимущества языка C++:

- скорость работы ресурсоемких алгоритмов превосходит Java до 3 раз;
- количество оперативной памяти, требуемое для работы приложения, намного меньше;
- reverse-engineering значительно сложнее, чем для языка Java.

Как видно из приведенных выше результатов анализа, язык программирования Java отличается более простой разработкой приложения, особенно если это касается UI и работы с периферией мобильного устройства. Однако если требуется реализовать ресурсоемкий алгоритм, не использующий периферию устройства, то язык C++ является более предпочтительным.

Выводы разделов "Защита от декомпиляции", "Производительность" и подразделов "Подход к использованию памяти" и "Скорость выполнения программ" справедливы как для языка C++, так и для языка Java, а заключения о производительности и реализации ресурсоемких алгоритмов актуальны для любого применения языков программирования C++ и Java.

Список литературы

1. **Collberg C., Thomborson C.** Watermarking, Tamper-Proofing, and Obfuscation — Tools for Software Protection: Technical Report 2000-03. Department of Computer Science. University of Arizona, 2000.
2. **Walle E.** Methodology and Applications of Program Code Obfuscation. University of Waterloo, 2001.
3. **Liang S.** The Java Native Interface Programmer's Guide and Specification. Sun Microsystems, Inc., 1999.
4. **Шилдт Г.** Полный справочник по Java SE 6 Edition. 7-е изд. М.: Вильямс, 2007. 1034 с.
5. **Memory** usage of Java objects: general guide. URL: http://www.javamex.com/tutorials/memory/object_memory_usage.shtml
6. **Размер Java**-объектов. URL: http://habrahabr.ru/post/134102/
7. **Dalheimer M. K.** A Comparison of Qt и Java. Linux development white paper, 2004.

УДК 378:004.77

А. А. Харламов, д-р техн. наук, вед. науч. сотр., e-mail: kharlamov@analyst.ru,
С. А. Смирнов, канд. пед. наук, стар. науч. сотр.,
Н. А. Сергиевский, стар. науч. сотр. e-mail: dereyly@gmail.com,
А. А. Жонин, стар. науч. сотр.,
Научно-исследовательский институт информационных технологий и телекоммуникаций "Информика", г. Москва

Интеллектуализация сервисов цифровых библиотек на основе самообучаемой системы классификации контента

Рассмотрены тенденции в развитии цифровых библиотек и их сервисов. Показано, что основное направление развития адаптивных сервисов цифровых библиотек связано с введением персонализации, которая улучшает качество их функций за счет подстройки к интересам пользователя. Предлагается подход к автоматической классификации на основе технологии для автоматического смыслового анализа текстов TextAnalyst как основание для формирования механизма персонализации. Описывается реализация программной системы на основе представленной технологии. Приводятся результаты экспериментального подтверждения эффективности использования технологии. Даны методические рекомендации по применению технологии автоматической классификации для интеллектуализации библиотечных сервисов.

Ключевые слова: *цифровые библиотеки, адаптивные сервисы, персонализация, автоматическая классификация, программная система, экспериментальная проверка, методика*

Введение

Отличительной чертой сегодняшнего этапа развития информационного общества является представление информации и знаний не только в традиционной печатной, но и в электронной, цифровой форме. Это позволяет принципиально по-иному создавать, хранить, организовывать доступ и использовать информацию в любой ее форме. Современные информационно-телекоммуникационные технологии привели к тому, что большое число новых информационных ресурсов сразу создается в электронном виде. Формируется новый класс информационных систем, предназначенных для управления электронными информационными ресурсами — цифровые библиотеки.

Цифровые библиотеки [1] — это фонды информации, которые при помощи разных технологий предоставляют сообществам пользователей информационные услуги. Фонды информации могут быть научными, деловыми и кадровыми. Информация в них может быть представлена в виде цифровых текстов, изображений, аудио- и видеоданных. Эта информация может представлять собой материалы, характеризующие производственную, научно-техническую деятельности, деловые документы, которые получены в цифровой форме. Предоставляемые библиотеками информационные услуги могут быть самыми разнообразными, от операций над контентом до управления правами на информацию. Они могут предоставляться как отдель-

ным пользователям, так и целым пользовательским сообществам. Совместное использование информации пользователями и их сотрудничество на основе представляемых библиотеками услуг стали важными элементами сферы общественных отношений.

Под общим названием "Цифровая библиотека" сегодня фигурируют следующие объекты.

- Архивы цифрового контента — хранилища переведенной в цифровую форму информации, снабженные минимальными интерфейсами доступа к этой информации, при этом не всегда даже сетевыми интерфейсами. Электронной библиотекой может называться DVD-диск вместе с прилагаемым программным обеспечением для доступа к цифровому контенту, организованному в виде файловой системы на этом диске.
- Набор программного обеспечения, реализующего основные функции управления цифровым контентом и организацию интерфейсов доступа к этому контенту.
- Системы сетевых сервисов, предоставляющих доступ к цифровому контенту, объединенные единой системой управления этим доступом.
- Некоторые организации, которые берут на себя ответственность не только за исполнение функций управления цифровым контентом и предоставления к нему доступа всем заинтересованным лицам, но и ответственность за соблюдение в процессе исполнения этих функций соответствия последних текущему законодательству в части обеспечения авторского права, приватности персональной информации, фильтрации паразитного трафика, сетевой безопасности клиента и др.

Новое поколение цифровых библиотек является разнородным по нескольким измерениям. Сами фонды становятся все более разнообразными в отношении их создателей, контента, носителей и обслуживаемых сообществ пользователей. Диапазон типов библиотек расширяется, охватывая долгосрочные личные цифровые библиотеки, а также цифровые библиотеки, которые обслуживают конкретные организации, нужды образования и культурного наследия, библиотеки, которые являются разными по надежности, авторитетности, степени новизны и качеству. Пользовательские сообщества становятся разнородными по их интересам, образовательному и интеллектуальному уровню, жизненному опыту, уровню квалификации пользователей — от новичков до экспертов в конкретных предметных областях. Растущее многообразие цифровых библиотек, сообществ пользователей и методов применения информации требует, чтобы следующее поколение библиотек стало более эффективным в предоставлении информации, которая будет адаптироваться к фоновым знаниям лица, его квалификации, задачам и предполагаемому использованию информации.

Разработки европейского сообщества в области цифровых библиотек, проводившиеся в рамках проектов Рамочных Программ Еврокомиссии, показали, что в Европе основное внимание уделяется решению вопросов, направленных на создание условий для более удобной работы пользователя с контентом. Основные функции, разработанные в этих проектах, касались накопления, конвертации, хранения и визуализации контента. В основном разрабатывались такие функции, как архивирование памяти сообществ (проект ARCOMEM), что позволяет максимально использовать для поиска, отбора и долговременного сохранения знания и опыт широких народных масс, представленные в социальных сетях, делает их более доступными современным и будущим пользователям. Создавались функциональные возможности доступа к аудиовизуальным архивам (AXES), которые позволяли пользователям найти новые способы взаимодействия с аудиовизуальными библиотеками, помогали пользователям в поиске, просмотре, навигации и пополнении архивов. Была реализована функция создания сетевого партнерства исследователей, центров культурного наследия, малых и средних предприятий сектора творческих индустрий (проект DigiBIC), которая сделала более легкими процессы распространения информации, обучения и оптимизации использования результатов ранее реализованных проектов. Разработанная в рамках этих проектов функция персонализированного доступа к пространствам культурного наследия (проект PATHS) обеспечила инновационный персонализированный доступ к коллекциям по культурному наследию и помощь пользователю в поиске и использовании знаний. Функция преобразования цифровых объектов, хранящихся на устаревших компьютерных носителях (проект KEEP), упростила их использование в современных устройствах путем создания портативных эмуляторов для точных преобразований статических и динамических цифровых объектов. Функция 3D-моделирования и создания виртуальной реальности для быстрой и экономичной реконструкции, визуализации и интерактивного использования городской среды (проект V-City) стала основой для крупномасштабных геопространственных библиотек. Функция архивирования живого Web (проект LiWA), которая позволяет охватывать информационные ресурсы из самых разнообразных источников, позволила улучшить процесс архивирования и сохранения аутентичности ресурсов, обеспечила возможности долговременной интерпретации контента.

Вместе с тем следует отметить, что цифровых библиотек становится все больше, их контент и предоставляемые услуги становятся все более разнообразными, а их постоянные пользователи все более опытными в компьютерных технологиях, поэтому от цифровых библиотек следует ожидать все более сложных услуг. Обычной функцией любой цифровой библиотеки является поиск. Однако с усложнением запросов пользователей и с увеличением объема информации, которой управляют цифровые библиотеки, растет дискомфорт пользователей и появляются новые потребности.

Цифровые библиотеки должны перейти от пассивного и лишь немного адаптированного по отношению к своим пользователям поведения к тому, чтобы работая на упреждение более широко предлагать и адаптировать информацию для отдельных пользователей и

их сообществ, поддерживать такие сообщества в их стремлении собирать, структурировать и совместно использовать знания. Есть все основания полагать, что цифровые библиотеки, которые не ориентированы на персональные интересы отдельных пользователей и/или сообществ, будут в дальнейшем позиционировать как не выполняющие своих обязательств по предложению наилучшего из возможных сервиса.

Поскольку компьютеры стали неотъемлемым элементом бизнеса, образования и персональной деятельности, личные цифровые библиотеки, рассчитанные на длительный период, становятся обычным делом. Информация, которая накапливается за время жизни компьютера, составляет личную цифровую библиотеку. Она делает каждого из нас и пользователем, и создателем цифровых библиотек. Люди всегда будут долго хранить свои картинки, музыку, образовательные и профессиональные материалы, личную и другую информацию. Вместе с тем их потребности, способности и вычислительные платформы будут меняться. Как следствие, будет востребовано долгосрочное моделирование интересов человека, выражающего свои предпочтения, обладающего определенными знаниями, ставящего себе какие-то цели и встроенного в социальные сети. Такие модели помогут человеку управлять своими личными цифровыми библиотеками на протяжении всего времени их использования. При этом такая информация не должна быть ограничена временем работы конкретных систем, последние будут часто меняться. Информация, которую человек собирает на протяжении всей жизни, то, как он организует ее, задачи, для которых она используется, и люди, с которыми она совместно используется, — все это ярко характеризует человека.

Первое поколение цифровых библиотек было создано для людей, информационные потребности которых были четки и хорошо согласовались с цифровыми ресурсами этих библиотек. Это было относительно однородное сообщество хорошо осведомленных пользователей с относительно точно описанными информационными потребностями. Эти характеристики ограничивали возможность использования цифровой библиотеки более широким сообществом.

Персонализация призвана сделать более разнородное заполнение цифровых библиотек доступным для все более разнохарактерного сообщества пользователей. Адаптивные сервисы следующего поколения цифровых библиотек должны обеспечить широкий диапазон персонифицированных услуг, которые будут поддерживать деятельность большого круга пользователей.

Рабочая группа DELOS/NSF [2] определяет персонализацию как способ, при помощи которого информация и услуги могут быть адаптированы чтобы удовлетворять конкретные потребности отдельного пользователя или некоторого сообщества. Такая цель достигается путем адаптации представления, контента и/или услуг к задачам пользователя, его жизненному опыту, предыстории, его оборудованию, информационным потребностям, местоположению, в общем — к контексту пользователя. Персонализация может направляться самим пользователем: он непосредственно вызывает и поддерживает процесс персонализации путем ввода конкретных данных. Примерами могут служить системы "MyYahoo!" и "MovieLens", в которых пользователь явно инициирует процесс и задает образец чтобы управлять персонализацией. Персонализация может также быть полностью автоматической, когда система следит за деятельностью пользователя и по используемой им информации адаптирует персонализированным способом некоторый аспект системы. Эти два примера управляемой пользователем и автоматической персонализации лежат на разных "концах технологического спектра". При этом многие инструментальные средства персонализации содержат элементы обоих подходов.

Ранние исследования по персонализации цифровых библиотек использовали простые модели пользовательских интересов, с их помощью выдавались лишь отдельные рекомендации. Будущие цифровые библиотеки должны использовать разные пользовательские модели, включая среду окружения человека, его знания, задачи, общественную деятельность и предпочтения. Полная информатизация потребует, чтобы цифровые библиотеки адаптировались к различным параметрам, связанным с контекстом работы человека. Для поддержания сообщества пользователей требуется, чтобы к отдельным пользовательским моделям были добавлены модели групп и сообществ.

Цифровые библиотеки могут быть персонализированы многими различными способами, позволяющими включить в систему большое число различных типов пользователей и их целей, а также видов решаемых ими задач [3]. Персонализация может быть основана на следующих типах характеристик:

- человека как отдельного пользователя или как члена некоторой группы (например, на знаниях или мотивациях);
- ресурсов информации или документов (например, класс материалов, их возраст и подлинность);
- полученного результата (например, новизна и точность), которые связаны с носителем или каналом (например, электронный помощник — карманный компьютер в противоположность сотовому телефону или стационарному компьютеру), с выполняемой задачей и окружением, в которое пользователь погружен.

Этот перечень характеристик относится к краткосрочной персонализации. Он не исчерпывающий, но выдвигает на первый план соотношения между самыми важными компонентами, а именно людьми, ресурсами и воспринятыми результатами. Перечисленные подходы к персонализации иллюстрируют богатство доступных типов данных, которые должны служить "рычагами" для ее управления.

В настоящее время персонализация сдерживается ограниченными возможностями моделирования пользователей, которое отражает слишком упрощенное их представление, включая поведения при поиске информации. Текущие пользовательские модели привлекают

лишь ограниченный набор параметров, в то время как люди, их рабочие должности и рабочие места являются намного более сложными для формального описания.

Пользовательские модели должны быть гибкими и динамичными, так как их информационные элементы должны изменяться во времени и пространстве, и, соответственно, должны меняться и сами модели. Можно предвидеть широкий доступ к цифровым библиотекам, в результате чего они станут повсеместными, мобильными, переносимыми и адаптивными. Тогда сходимость пространства личной информации отдельного пользователя с глобальным информационным пространством будет способствовать созданию действительно личных цифровых библиотек. Наконец, пользовательские модели должны включить социальные характеристики человеческого поведения и предпочтений, благодаря которым членство в социальной и рабочей группах могут влиять на нужды и потребности человека.

Таким образом, системы персонализации подразделяются на серверные (групповые) и клиентские (индивидуальные). Серверная персонализация естественна для построения моделей групп пользователей и их сообществ. Она может быть хорошо интегрирована с контентом и услугами, которые предлагает цифровая библиотека. Клиентская персонализация естественна для построения более подробной формальной модели отдельного пользователя по множеству его задач, транзакций и по всему периоду ее использования. Такой подход дает бóльший контроль над тем, как получаются персональные данные и какие позиции их могут быть открыты. Например, данные для обучения, имеющиеся у клиента, значительно отличаются от тех, что фактически хранятся на сервере. Этот факт требует того, чтобы клиент намного глубже понимал значение пользовательского взаимодействия с разными информационными услугами и ресурсами.

Баланс между пользовательской персонализацией и персонализацией, основанной на сообществе, будет меняться для отдельного пользователя и конкретного ресурса или задачи. Если человек впервые обращается в цифровую библиотеку, то персонализация может быть выполнена на основе общности между независимой от библиотеки моделью отдельного пользователя и подобными отдельными пользователями, которые взаимодействовали с этой цифровой библиотекой в прошлом. Поскольку у отдельного пользователя больше опыта общения с цифровой библиотекой, то со временем баланс между пользовательской персонализацией и персонализацией, основанной на сообществе, сдвинется, и вклад, вносимый отдельным пользователем в сообщество и в модель сообщества, увеличится.

Адаптивные сервисы

Адаптивные сервисы, в соответствии с делением персонализации на индивидуальную и групповую, также делятся на адаптируемые к потребностям отдельного пользователя и адаптируемые к потребностям целых групп пользователей.

Первые базируются на портрете индивидуального пользователя, вторые используют обобщенные характеристики социальных групп (например, по возрасту, полу, социальному положению и другим признакам).

Персональная цифровая библиотека является основой для формирования индивидуальных портретов пользователей, которые создаются в течение всей жизни пользователя в виде структурированного множества текстов, накопленного в персональной библиотеке. Структура этой коллекции включает разделение на учебную, профессиональную, научную и развлекательную части.

Цифровая библиотека, которая используется социальными группами, может быть основой для формирования обобщенного портрета социальной группы. С одной стороны, обобщенный портрет социальной группы может быть сформирован усреднением из персональных портретов ее членов. С другой стороны, обобщенный портрет может быть сформирован на основе корпусов текстов отдельных социальных групп путем формального учета и регистрации пользователей цифровой библиотеки.

Адаптивные сервисы цифровых хранилищ и библиотек базируются на одном из основных механизмов обеспечения автоматической обработки текстовой информации — механизме автоматической классификации текстов.

Автоматическая классификация текстов — хорошо разработанный раздел научного направления "Искусственный интеллект". Основные подходы к ней укладываются в рамки статистической и лингвистической обработки.

Как и в других областях обработки текстовой информации, эти два подхода противопоставляются по качеству/скорости обработки. Лингвистический подход дает высокое качество, но проигрывает в скорости, статистический подход уступает лингвистическому в качестве, но обладает большей скоростью. Поскольку речь, как правило, идет об обработке в реальном времени больших массивов текстовой информации, естественно, предпочтение отдается скорости.

Подход к классификации текстов на основе технологии смыслового автоматического анализа текстов TextAnalyst

С учетом изложенного ранее можно сказать, что в основе построения адаптивных сервисов, базирующихся на механизмах классификации, лежит статистический по своей природе нейросетевой подход к автоматическому смысловому анализу текстовой информации TextAnalyst [4]. В основных своих чертах он сводится к автоматическому формированию статистического сетевого портрета текста (корпуса текстов), который в этом подходе эквивалентен индексу, а также тезаурусу текста других подходов. Из текста автоматически извлекаются ключевые понятия (слова и устойчивые словосочетания), в их смысловых (в терминах ассоциативности) взаимосвязях, нормированные весовыми характеристиками, которые ранжируют по-

нятия и связи в терминах их смысловой (структурной) значимости в тексте. Построенные таким образом смысловые портреты текстов могут быть использованы для их смыслового сравнения путем вычисления степени пересечения сетей этих текстов.

В качестве одного из текстов может быть целый корпус текстов — рубрика каталога. Тогда смысловое сравнение позволяет автоматически классифицировать тексты по рубрикам.

Заранее сформированные и формируемые далее во времени текстовые выборки, характеризующие отдельных пользователей (в случае индивидуальной персонализации), а также характеризующие социальные группы (в случае групповой персонализации), и являются рубриками с ранжированными предпочтениями, которые позволяют классифицировать входные тексты, предназначенные для пользователя.

Сравнение семантических сетей текстов, в отличие от сравнения индексов, позволяет выявить в тексте важнейшие ключевые понятия в их взаимосвязях, которые и характеризуют основные предпочтения, заключенные в тексте. В отличие от подходов на основе тезаурусов, которые формируются вручную, резко уменьшается время на обработку информации. Хотя, если смотреть более глубоко, упомянутая семантическая сеть текста по существу является его тезаурусом, а именно — множеством ключевых понятий, взаимоувязанных в соответствии с их смысловыми связями в этом тексте, если дополнить семантическую сеть ссылками на предложения (как словарные статьи), содержащие ключевые понятия сети.

Дополнительным положительным качеством предложенного подхода является независимость качества обработки от принадлежности к предметной области, а также независимость от языка (для европейской группы языков).

Программная система для структурного анализа текстов TextAnalyst

На основе нейросетевой технологии Научно-производственным инновационным центром "Микросистемы" (г. Москва) было разработано семейство программных продуктов для автоматического смыслового анализа текстовой информации TextAnalyst [5].

Разработанная система обработки текстовой информации основана на использовании структурных свойств языка и текста. Такие свойства могут быть выявлены с помощью статистического анализа, реализованного на основе иерархических структур из искусственных нейронных сетей на основе нейроподобных элементов с временной суммацией [4]. Данная система реализует: автоматическое формирование описания семантики предметной области текста; функции организации текстовой базы в гипертекстовую структуру; функции автоматического реферирования, кластеризации и классификации текстов; функцию смыслового поиска.

Иерархические структуры из искусственных нейронных сетей упомянутого выше типа являются удобным инструментом для выявления структурных свойств текстовой информации. Использование указанного инструментария позволяет автоматически, на основе анализа статистики слов и их связей в тексте, реконструировать внутреннюю структуру текста.

Статистический анализ выявляет наиболее часто встречающиеся элементы текста — слова или устойчивые словосочетания. Важной особенностью используемого подхода является возможность автоматически устанавливать взаимосвязи между выявленными элементами текста. При выявлении связей учитывается статистика попарного появления слов во фрагментах исследуемого текстового материала. Далее статистические показатели пересчитываются в семантические с помощью алгоритмов, подобных алгоритму перерасчета весов в сетях Хопфилда [6]. Идея подобных алгоритмов заключается в том, что при расчете какой-то характеристики элемента сети учитываются подобные характеристики элементов, с ним связанных, а также принимаются во внимание численные характеристики силы связей. После пересчета статистических характеристик в семантические, понятия, которые мало соответствуют анализируемой предметной области, получают малый вес, а наиболее представительные наделяются высокими показателями. Полученная семантическая сеть позволяет проводить различные виды анализа текстовой информации. Сеть отражает внутреннюю структуру текста, значимость выделенных понятий, а также показывает степень связанности понятий в тексте. Такое представление текста получается полностью автоматически.

Семантические веса элементов сети используются при расчете близости (релевантности) фрагментов текста к запросу пользователя при поиске информации в тексте. На их основе возможно выделение наиболее информативных участков текста. Использование ассоциативных связей элементов сети позволяет расширять поле поиска информации. Ответ на запрос пользователя в этом случае может содержать информацию, явно не указанную в запросе, однако связанную с ней по смыслу.

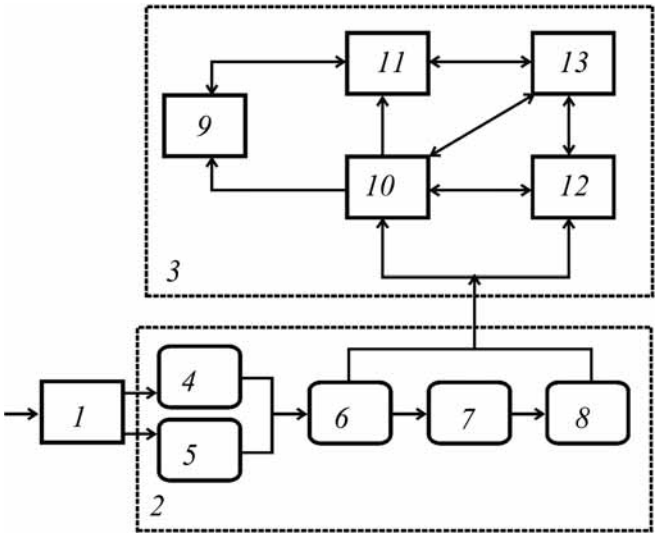
Архитектура системы

Программная система для смыслового анализа текстовой информации реализована как инструментарий для автоматического формирования баз знаний на основе множества естественно-языковых текстов. Ядро системы выполнено как программный компонент (*inproc server*), соответствующий спецификации Component Object Model (COM) фирмы Microsoft.

Ядро системы реализует следующие функции:

- нормализацию грамматических форм слов и вариаций словосочетаний;
- автоматическое выделение базовых понятий текста (слов и словосочетаний) и их взаимосвязей с вычислением их относительной значимости;
- формирование представления семантики текста (множества текстов) в форме семантической сети.

Система анализа текстов (см. рисунок) содержит блок первичной обработки (*1*), лингвистический и семантический процессоры. Лингвистический процес-



Система анализа текстов:
1 — блок первичной обработки; *2* — лингвистический процессор; *3* — семантический процессор

сор (*2*) состоит из словарей: слов разделителей (*4*), служебных слов (*5*), общеупотребимых слов (*6*), а также флективных (*7*) и корневых морфем (*8*). Семантически процессор (*3*), в свою очередь, содержит: блок отсылок в текст (*9*), блок формирования семантической сети (*10*), блок хранения семантической сети (*11*), блок выделения понятий (*12*) и блок управления (*13*).

Блок первичной обработки. Задачами этого блока являются извлечение текста из файла (входного потока данных) и подготовка его к обработке в лингвистическом процессоре. Подготовка текста заключается в очистке его от символов, неизвестных лингвистическому процессору, а также в корректной обработке таких единиц текста как аббревиатуры, инициалы, заголовки, адреса, номера, даты, указатели времени.

Лингвистический процессор. Лингвистический процессор осуществляет предобработку входного текста (последовательности символов в определенной кодировке) на основе априорных лингвистических знаний, общих для выбранного языка (в настоящее время поддерживаются несколько европейских языков помимо русского и английского), и выполняет:

- сегментацию предложений текста на основе знаков пунктуации и специальных грамматических слов, их фильтрацию;
- нормализацию слов и словосочетаний — фильтрацию флексий (окончаний) с сохранением только корневых слов;
- фильтрацию в тексте семантически несущественных, вспомогательных слов, удаляя предлоги, числительные и самые общеупотребимые слова с широким значением;
- маркировку общеупотребимых слов.

Сегментация предложений позволяет разбить текст на участки, которые могут содержать терминологические словосочетания предметной области и, тем самым, избежать выделения неадекватных словосочетаний на стыках таких участков.

В результате предобработки семантически близкие словосочетания приводятся к одинаковой форме (нормализуются). Маркировка общеупотребимых слов необходима в целях исключения их выделения как самостоятельных терминов при дальнейшем анализе.

База общих языковых знаний лингвистического процессора содержит словари, по одному для реализации каждой из четырех функции: словарь слов-разделителей предложения; словарь вспомогательных слов; словарь флексий; словарь общеупотребимых слов.

Блок выделения понятий. Блок выделения ключевых понятий предметной области (слов и словосочетаний) создан на базе программной модели иерархических структур из искусственных нейронных сетей. Он реализует алгоритмы автоматического формирования частотного словаря текста.

Число уровней в иерархической структуре определяет априорно заданную максимально допустимую длину понятия предметной области и равняется двадцати.

На первом уровне иерархической структуры представлен словарь двухбуквенных специальных слов предметной области. К таким относятся слова, пропущенные через все фильтры лингвистического процессора и не отнесенные к общеупотребимым, а также двухбуквенные сочетания из слов этого словаря. Там же хранятся двухбуквенные слова общеупотребимой лексики, входящие в устойчивые словосочетания, а также их начальные двухбуквенные фрагменты. На втором уровне иерархической структуры хранятся словари трехбуквенных слов и сочетаний букв из словарей специальных и общеупотребимых слов, которые встречались в тексте, в виде индексов элементов соответствующих словарей первого уровня, дополненных еще одной буквой. На последующих уровнях представление информации полностью однородно — на каждом уровне хранятся индексы элементов более низкого уровня, дополненные одной буквой.

В процессе формирования представления информации в иерархической структуре подсчитывается частота встречаемости каждого сочетания букв в соответствующих элементах. Частота слов (сочетаний букв, не имеющих продолжения на следующем уровне) используется для последующего анализа.

Сформированное таким образом представление лексики текста подвергается затем пороговому преобразованию по частоте встречаемости. Порог отражает степень детализации описания текста. В процессе статистического анализа в иерархической структуре выделяются устойчивые термины и терминологические словосочетания, которые служат далее в качестве элементов для построения семантической сети. При этом общеупотребимые слова, а также словосочетания, содержащие только общеупотребимые слова, опускаются.

Блок формирования семантической сети. Блок формирования семантической сети реализован как база данных, в которой представляются семантические

связи понятий предметной области. Поскольку типы семантических связей [7] в системе не определяются, такие связи являются ассоциативными.

В качестве критерия для определения наличия семантической связи между парой понятий используется частота их совместной встречаемости в одном предложении. Превышение такой частотой некоторого порога позволяет констатировать наличие между понятиями ассоциативной (семантической) связи, а совместные вхождения понятий в предложения с частотой меньше этого порога считаются просто случайными.

Элементы семантической (ассоциативной) сети и их связи имеют числовые характеристики, отражающие их относительный вес в данной предметной области — семантический вес. При достаточно представительном множестве текстов, описывающих предметную область, значения частот встречаемости понятий действительно отражают соответствующие семантические (субъективно оцениваемые) веса. Однако для небольших обучающих выборок, в частности, при анализе отдельного текста, не все частотные характеристики соответствуют действительным семантическим весам — важности понятий в тексте. Для более точной оценки семантических весов понятий используются веса всех связанных с ними понятий, т. е. веса целого "семантического сгущения". В результате такого анализа наибольший вес приобретают понятия, обладающие мощными связями и находящиеся как бы в центре "семантических сгущений".

Основные функции системы TextAnalyst

На основе результатов работы модуля индексации реализованы следующие функции обработки текстовой информации: формирования гипертекстовой структуры; навигации по базе знаний; формирования тематического дерева; реферирования текстов; автоматической кластеризации множества текстов; сравнения текстов (автоматической классификации текстов); функция формирования ответа на смысловой запрос пользователя (формирования тематического реферата).

После формирования семантической сети исходный текст, объединенный гиперссылками с семантической сетью, становится гипертекстовой структурой. Семантическая сеть в этом случае оказывается удобным средством навигации по тексту. Она позволяет исследовать основную структуру текста, переходя от понятия к понятию по ассоциативным связям. Пользуясь гиперссылками пользователь может быстро найти множество предложений текста, содержащих эти понятия. С помощью тех же гиперссылок он может перейти от любого предложения непосредственно к его контексту в тексте. С этой же целью пользователь может пользоваться минимальным древовидным подграфом семантической сети — тематическим деревом. В нем оказываются иерархически представленными основные и соподчиненные понятия сети, при этом понятия нижнего уровня объясняют содержание понятий более высокого уровня. Тематическим деревом можно также пользоваться для навигации по базе знаний — оно напоминает оглавление текста.

Семантическая сеть с числовыми характеристиками ее компонентов — понятий и их связей — позволяет вычислить вес каждого предложения в тексте. Множество предложений текста, выбранных в порядке их появления в тексте, вес которых превысил некоторый пороговый уровень, можно считать рефератом текста.

Семантическая сеть исследуемого текста (или группы текстов) может быть разбита на подсети удалением из нее слабых связей. Каждая такая подсеть группируется вокруг некоторого понятия с максимальным весом в данной подсети. Это понятие обозначает тему части текста или отдельных текстов, которые оказываются сгруппированными в данной подсети. Такая автоматическая кластеризация позволяет разбить множество текстов на рубрики, а также визуализировать динамику развития этих рубрик во времени.

Используя числовые характеристики семантической сети, можно сравнивать сети двух текстов с точки зрения вычисления их пересечения (общей части). Таким образом, можно сравнивать степень совпадения текстов по смыслу. Если в качестве одного из текстов берется целая рубрика, то существует возможность оценить степень принадлежности исходного текста к данной рубрике, т. е. автоматически классифицировать тексты.

Система для смыслового анализа текстов позволяет реализовать также смысловой поиск (сформировать тематический реферат). Функция смыслового поиска, основываясь на ассоциативном иерархическом представлении содержания информации в базе, на функциях кластеризации и классификации, осуществляет выборку информации, соответствующей запросу пользователя, и структурирует ее в соответствии с близостью к запросу.

Смысловой поиск, используя ассоциации, способен выдавать пользователю явно не указанную в тексте запроса информацию, но связанную с ней по смыслу. Использование такого подхода ведет не к увеличению выдаваемой пользователю информации, а к ее тщательному отбору и анализу по главному критерию — смысловой близости к запросу.

Классификатор для адаптивных сервисов, построенный на основе метода опорных векторов

В направлении классификации текстов большое распространение получили подходы на основе метода опорных векторов [8]. При работе с текстами чаще всего используют линейные методы. Это обучаемые методы, которые предназначены для настройки вектора весов в процессе обучения таким образом, чтобы максимизировать корреляционную меру (произведение вектора признаков и вектора весов) для объектов класса, минимизируя при этом корреляционную меру для объектов, не принадлежащих данному классу. Для построения вектора признаков используется частот-

ный портрет текста, полученный с помощью технологии TextAnalyst. Частотный портрет, представляющий собой перечень слов с их частотами встречаемости, проецируется на словарь, составленный из обучающего корпуса текстов. В итоге получается разреженный вектор признаков, в ненулевых элементах которого содержатся нормированные числа от 0 до 1 — частоты встречаемости.

Одним из самых современных и удобных линейных классификаторов на данный момент является liblinear [9], который позволяет быстро обучаться на данных большой размерности за счет двухкритериальной оптимизации по методу покоординатного спуска.

Отметим особенность используемого метода опорных векторов.

Задачей линейного метода опорных векторов является поиск оптимальной гиперплоскости, разделяющей пространство на два класса:

$$w \cdot x - b = 0,$$

где w — это перпендикуляр к разделяющей гиперплоскости, b — смещение, подбор этих параметров является целью оптимизационной задачи:

$$\frac{1}{2} w^{\mathrm{T}} w + c \sum_{i=1}^{l} \xi(w; x_i; y_i) \to \min_w .$$

Первый член этой оптимизационной задачи максимизирует ширину разделяющей полосы, второй член за счет функции потерь $\xi(w; x_i; y_i)$ ставит в соответствие входным данным x_i выходной класс y_i; c — параметр настройки метода, который позволяет регулировать отношение между максимизацией ширины разделяющей полосы и минимизацией суммарной ошибки.

В рамках решения категоризационной задачи в liblinear применяем следующую оптимизационную стратегию:

$$\|w\|_1 + c \sum_{i=1}^{l} (\max(0,1 - y_i w^{\mathrm{T}} x_i))^2 \to \min_w .$$

Данная стратегия отдельно оптимизирует регуляризационный член $\|w\|_1$ в L1, а функцию потерь $(\max(0,1 - y_i w^{\mathrm{T}} x_i))^2$ в L2. Такой подход показал наиболее высокий результат на выборке DMOZ [10].

Формирование вектора признаков для алгоритма линейной классификации

В случае линейной классификации качество классификации зависит от метода формирования вектора признаков. В данной работе используется вектор признаков, полученный на основе семантического портрета текста [4]. Под семантическим портретом текста подразумевается перечень ключевых понятий (понятие может быть как словом, так и словосочетанием) с их семантическими весами, полученными с помощью программы TextAnalyst SDK. Семантические веса понятий являются их нормализованными характеристиками, выделяющими основные понятия, и в дополнительной перенормировке (типа tf-idf) не нуждаются. На основании полученных на всем корпусе текстов понятий формируется словарь предметной области. После проецирования семантического портрета на этот словарь получаем разреженный вектор признаков, в ненулевых элементах которого находятся семантические веса понятий. Этот вектор признаков теоретически представляет более стабильную и универсальную структуру для классификации текстов, что подтверждается результатами проведенных экспериментов.

Эксперимент

Эксперимент проводился для сравнения перспективных подходов к классификации текстов, в том числе на основе метода опорных векторов и на основе латентно-семантического анализа. В обоих случаях ключевую роль играет замена частотного портрета текста на его семантический портрет, полученный с помощью технологии TextAnalyst.

Основной задачей в экспериментальной части было определение оптимального числа слов в словаре, достаточного для описания классов в выборке. Рассчитывалась кроссвалидационная точность классификации текстов выборки DMOZ [10]. Предварительная обработка текстов с помощью технологии TextAnalyst использовалась для выявления в текстах ключевых понятий (слов и устойчивых словосочетаний) и их семантических весов. В качестве понятий рассматривались слова и словосочетания, в которых главным словом выступает существительное. После обработки обучающей выборки было построено два словаря. Первый словарь содержал только слова, второй — слова и словосочетания не длиннее трех слов. Объем первого словаря составлял 47 000 слов, второго — 124 000 слов. Базовый результат — точность классификации 94 % — был получен с использованием словаря, учитывающего словосочетания.

Далее были исследованы методы по сокращению словаря. Метод на основе выбора наибольших значений для tf-idf (в нашем случае в качестве tf выступал семантический вес слова, полученный обработкой с помощью технологии TextAnalyst) на словаре 5000 слов показал точность 90 %. Второй метод — на основе латентно-семантического анализа с векторами признаков размерностью 100 и 200. Он показал (на линейном SVM — Support Vector Machine) худший результат (примерно 47 и 61 % соответственно), чем на полном словаре (124 000).

Методика классификации пользователей в адаптивных сервисах

Использование технологии автоматического смыслового анализа текстов TextAnalyst позволяет формировать адаптируемые сервисы цифровых хранилищ следующим образом.

Интеллектуальные адаптируемые сервисы, а именно поисковые, аналитические, развлекательные и иные сервисы для контента различной направленности (профессиональная, художественная литература, научно-техническая информация, учебная и детская литература и т. п.), различной тематики и жанров, могут быть напрямую реализованы с использованием упомянутой технологии. Для этого формируются корпуса текстов, характеризующие пользователей (как персональных, так и групповых). При этом множество входных текстов, полученных, например, в результате обработки запроса пользователя поисковиком, классифицируется в соответствии с рубриками персонального портрета. Ранжирование классифицированных текстов и определяет степень близости входных текстов к предпочтениям пользователя.

Интеллектуальные адаптируемые сервисы для различных категорий пользователей (по социальным признакам) реализуются несколько сложней. При построении семантических сетей персональных портретов, характеризующих пользователей по социальным признакам, отнесение к предметной области (физика, математика, искусство) должно быть элиминировано. Для этого из семантических сетей удаляются ключевые понятия, относящие тексты к конкретным предметным областям. Оставшаяся часть сети характеризует пользователя текста по иным, отличным от предметной области, признакам, например, социальным. Этот факт означает, что сначала строится общая сеть, затем из нее удаляется подсеть предметной области, а после этого оставшаяся часть сравнивается с такими же частями рубрик персонального портрета пользователя. В этом случае ранжирование расклассифицированных текстов определяет степень близости входных текстов к предпочтениям пользователя.

Заключение

Интеллектуальные адаптируемые сервисы, включая поисковые, аналитические, развлекательные и иные сервисы для контента различной направленности, различной тематики и жанров, различных категорий пользователей, получают все большее распространение.

Проследив тенденции развития сервисов цифровых библиотек, можно заметить, что наряду с решением вопросов удобства работы пользователя с контентом, в том числе его накоплением, конвертацией, хранением и визуализацией, при их разработке решаются вопросы адаптации сервисов под пользователя. При формировании адаптивных сервисов необходимо учитывать контекст информационного пространства пользователя, включающий:

- когнитивные способности, например, манеру учиться, восприятие;
- индивидуальные различия, например, опыт, образование, возраст, пол;
- образцы поведения и предысторию отдельного пользователя/группы;
- предметные области, например, инженерно-технические отрасли, искусство, здоровье;
- рабочие задачи, например, написание сочинений, выбор кинофильма, планирование отпуска;
- окружение на работе, например, университет, больница, офис, дом;

а также как все перечисленное меняется со временем.

Решение этих проблем лежит в русле создания механизмов автоматической обработки контента. Причем основное направление исследований в области интеллектуализации сервисов цифровых библиотек должно быть сосредоточено на вопросах персонализации.

Предлагаемый подход автоматического разделения контента на категории на основе технологии Text-Analyst может быть использован для разделения контента различной тематики и жанров. Предлагаемый подход позволяет также автоматически более точно разделить множество текстов для различных категорий пользователей по социальным признакам.

Список литературы

1. **McDaniel B., Ryszard K. S.** Semantic Digital Library / Goals of semantic digital libraries. Springer, 2009. P. 79—80. 245 p.
2. **Smeaton A., Callan J.** Joint DELOS-NSF. Workshop on Personalization and Recommender Systems in Digital Libraries School of Computer Applications School of Computer Science. Dublin City University. URL: http://www.sigir.org/forum/S2001/DELOS.pdf
3. **Linckels S., Meinel C.** E-Librarian Service: User-Friendly Semantic Search in Digital Libraries / Design of natural language processing module. Springer, 2011. P. 118—119. 205 p.
4. **Харламов А. А.** Нейросетевая технология представления и обработки информации (естественное представление знаний). М.: Радиотехника, 2006. 89 с.
5. **Kharlamov**, 2012 URL: http://www.analyst.ru.
6. **Hopfield J. J.** Neural networks and physical systems with emergent collective computational abilities // Proceedings of the National Academy of Sciences. 1982. N 79. P. 2554—2558.
7. **Осипов Г. С.** Приобретение знаний интеллектуальными системами: Основы теории и технологии. М.: Наука. Физматлит, 1997. 112 с.
8. **Ермакова Л. М.** Методы классификации текстов и определения качества контента // Вестник Пермского университета. 2011. № 3 (7). С. 47—53. URL: http://vestnik.psu.ru/files/articles/242_58596.p
9. http://www.csie.ntu.edu.tw/~cjlin/liblinear/
10. http://www.dmoz.org/

УДК 004.05

В. В. Баранюк, канд. техн. наук, стар. науч. сотр., глав. науч. сотр.,
Н. Н. Тютюнников, канд. техн. наук, стар. науч. сотр., вед. науч. сотр.,
ФГУП "Центральный научно-исследовательский институт экономики, информатики и систем управления", г. Москва
e-mail: tutunnikov_nn@cniieisu.ru

Оценка качества электронных словарей и энциклопедий

Представлены подходы, которые использовали авторы для оценки качества электронных словарей и энциклопедий на основе методологии квалиметрии. Методика оценки включает этапы определения ее цели, выбора номенклатуры показателей качества, базового образца и методов определения значений показателей качества, определения значений показателей качества и оценки научно-технического уровня. Оценка проводилась для 54 электронных словарей и энциклопедий, разработанных в течение последних 10...15 лет различными компаниями.

Ключевые слова: *качество продукции, квалиметрия, электронные издания*

Введение

В соответствии с действующими нормативно-техническими документами [1] *автоматизированная система* [2] является организационно-технической системой, обеспечивающей выработку решений на основе автоматизации информационных процессов в различных сферах деятельности (управление, проектирование, производство и т. д.) или их сочетаниях. В процессе функционирования автоматизированная система представляет собой совокупность *комплекса средств автоматизации* [2], организационно-методических и технических документов и специалистов, использующих их в процессе своей профессиональной деятельности. В процессе проектирования автоматизированной системы или ее частей, в общем случае, разрабатывают девять видов ее обеспечения, представляющих собой совокупность методов, средств и мероприятий, реализующих деятельность по организации бесперебойного и эффективного функционирования автоматизированной системы. Проектные решения по всем видам обеспечения реализуются как программные, технические и информационные изделия в виде взаимоувязанной совокупности компонентов и комплексов, входящих в состав автоматизированной системы или ее частей с комплектом необходимых организационно-методических и эксплуатационных документов.

Практика разработки автоматизированных систем показала, что одними из наиболее важных являются следующие виды обеспечения, которым не всегда уделяется должное внимание:

— *информационное*, представляющее собой совокупность форм документов, классификаторов, нормативной базы и реализованных решений по объемам, размещению и формам существования информации, применяемой в автоматизированной системе при ее функционировании;

— *лингвистическое*, представляющее собой совокупность средств и правил для формализации естественного языка, используемых при общении пользователей и эксплуатационного персонала автоматизированной системы с комплексом средств автоматизации при функционировании автоматизированной системы [2].

Наибольший эффект от внедрения автоматизированной системы достигается лишь в том случае, если работающие с такой системой должностные лица и другие пользователи получают достоверную и актуальную информацию, необходимую им для принятия решений или выдачи управляющего воздействия на объект управления. Кроме того, создаваемые автоматизированные системы должны иметь информационную совместимость между собой в части, определенной их назначением, что приводит к необходимости унификации используемых ими информационных компонентов.

Одним из основных компонентов лингвистического обеспечения автоматизированной системы является система словарей, терминологическое и словарное наполнение которой реализуется в информационном обеспечении, а методы ее обработки — в программном. Сложность построения унифицированной системы словарей крупных автоматизированных систем состоит в том, что необходимо учитывать различные варианты организации словарных массивов и реализовывать типовые подходы их использования. Эффективным способом решения таких задач является использование научно обоснованных методов оценки качества изделий одинакового назначения с параллельным контролем качества создания новых изделий, учитывающих накопленный опыт в данной области знаний.

Методические основы оценки качества продукции

В России в последнее время для оценки качества продукции используют гармонизированные стандарты серий ГОСТ Р ИСО 9000 и ГОСТ Р ИСО 10000 "Системы качества", "Системы менеджмента качества", "Менеджмент качества", а также другие стандарты по различным видам продукции, например, ГОСТ Р ИСО/МЭК 9126—93 для оценки программной продукции. Для разработки *продукции военного* и *двойного назначения (применения)* действуют свои стандарты, устанавливающие более жесткие требования к разрабатываемым изделиям. К их числу относится ГОСТ 15467—79 [3], в соответствии с которым одним из главных показателей качества продукции является *технический уровень*. Для автоматизированных систем установлено понятие *научно-технического уровня* [2], используемое также в нормативных документах на создание изделий.

Отдельным направлением оценки качества продукции [4] является оценка научно-технического уровня продуктов и услуг, которые разрабатываются в рамках научно-исследовательских и опытно-конструкторских работ. При их проведении поддержание заданного (или высокого) уровня качества осуществляется различными *методами управления качеством* [5], необходимым условием применения которых является *контроль качества продукции* [6].

Один из основных способов контроля качества продукции основан на методологическом аппарате *квалиметрии* [3]. Термин "квалиметрия" впервые был использован в 1968 г. [7] для обозначения научной дисциплины, изучающей методологию и проблематику количественного оценивания качества объектов любой природы [8]. Для получения таких оценок качества продукции основателем квалиметрии Г. Г. Азгальдовым в 1973 г. были определены принципы квалиметрии [9].

Электронные словари и энциклопедии, как результат научно-технической деятельности, являются *информационной продукцией* [10]. В автоматизированной системе электронные словари и энциклопедии являются компонентом лингвистического обеспечения автоматизированной системы и могут разрабатываться как *информационное изделие* или *программное изделие в автоматизированной системе* [2]. По этой причине оценку качества электронных словарей и энциклопедий необходимо осуществлять с учетом нормативно-технических документов на автоматизированные системы (РД 50-492—84) и программные средства (ГОСТ 28195—89).

Для оценки качества электронных словарей и энциклопедий использована методика, основанная на принципах квалиметрии. Она включает следующие этапы:

- определение цели оценки качества;
- выбор номенклатуры показателей качества;
- выбор базового образца;
- выбор методов определения значений показателей качества;
- определение значений показателей качества;
- оценка уровня качества (научно-технического уровня).

Цель оценки качества

Оценка качества электронных словарей и энциклопедий осуществлялась в рамках такого направления квалиметрии как *проектная квалиметрия*, которая представляет собой совокупность процедур системного (комплексного) исследования качества и анализа условий и факторов, влияющих на эти процедуры в процессе проектирования техники [4]. В проектной квалиметрии решаются *обратные задачи квалиметрии*, исследующие качественные характеристики объектов в целях синтеза "пригодного" или "лучшего" по качеству нового объекта. Решение задачи должно обеспечить усовершенствование существующего объекта до получения пригодного (не хуже некоторого требуемого) или лучшего его качества либо синтез принципиально нового объекта, обладающего пригодным или лучшим качеством. Синтез, как правило, представляет собой итеративный процесс, в котором осуществляется последовательное усовершенствование качества объекта с циклическим повторением ряда этапов, связанных с оценкой уровня качества и принятием решения о соответствии результата требуемому значению. Процедура заканчивается после того, когда достигнута цель или когда приходят к выводу, что невозможно получить требуемое качество и необходимо искать другие пути решения задачи.

С точки зрения авторов, перспективным путем реализации системы словарей в лингвистическом обеспечении автоматизированной системы является ее создание из следующих элементов:

— терминологический фонд, комплектуемый на основе терминов и определений из нормативных документов, официально изданных энциклопедий и словарей, терминологических словарей других автоматизированных систем, а также других первоисточников;

— электронный словарь терминов (ЭСТ), являющийся центральной частью терминологического фонда и ведущийся на его основе;

— другие взаимосвязанные информационные фонды, хранящие в том числе документы-первоисточники терминологического фонда.

Такой подход, заключающийся в создании ЭСТ на основе сведений из терминологического фонда, позволяет систематизировать существующие понятия и соответствующие им предметные области по различным классификационным схемам. Кроме того, появляется возможность устранения противоречий толкования терминов в различных изданиях путем включения в ЭСТ определений, имеющих больший уровень значимости в соответствии с правовым статусом документа-первоисточника.

Такая агрегация терминов и определений в ЭСТ из различных словарей осуществляется не только для удовлетворения информационных потребностей должностных лиц и других пользователей автоматизированной системы разнообразной терминологией и другой справочной информацией. Она необходима и для обеспечения терминологического соответствия при представлении сведений в информационных базах автоматизированных систем и информационном обмене между ними. Эффективность такого подхода обусловлена тем, что любые изменения в документах-первоисточниках или словарях других автоматизированных систем влекут за собой автоматические изменения в ЭСТ. Это обстоятельство существенно сокращает затраты по его поддержанию в актуальном состоянии.

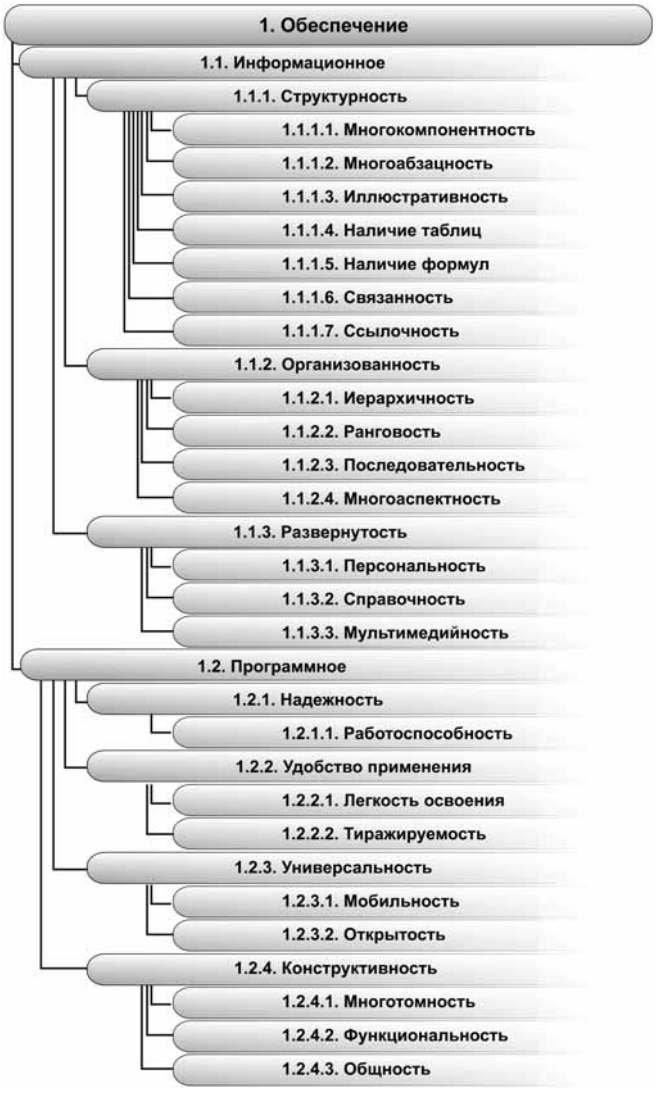
С учетом изложенных выше соображений, целью оценки качества электронных словарей и энциклопедий являлось выявление таких их качеств, реализация которых позволила бы добиться лучшего уровня качества ЭСТ в соответствии с его предназначением.

Номенклатура показателей качества

Номенклатура показателей качества для оценки научно-технического уровня электронных словарей и энциклопедий, представленная на рисунке, выбиралась исходя из требования, что электронное справочное издание может быть составной частью информационного и программного обеспечения автоматизированной системы. Такое требование повлекло за собой необходимость выделения двух групп основных комплексных показателей.

Система показателей качества имеет четыре уровня иерархии, верхний из которых фактически характеризует научно-технический уровень издания. Такое построение системы обусловлено тем, что в этом случае предоставляется возможность ее встраивания в полные системы оценки показателей качества автоматизированных систем, разрабатываемых на основе требований технического задания. Встраивание осуществляется как на уровне общесистемных требований (показатель "Обеспечение"), так и на уровне требований в видах обеспечения автоматизированной системы (показатели "Информационное" и "Программное").

При разработке показателей, отвечающих за информационную часть, прежде всего учитывались особенности построения базы данных ЭСТ и его информационного наполнения. Показатели программной части разрабатывались на основе действующих нор-



Номенклатура показателей качества для оценки научно-технического уровня электронных словарей и энциклопедий

мативных документов по оценке качества программных средств с учетом типовых функциональных возможностей, реализуемых в электронных справочных изданиях. Описание системы показателей качества представлено в табл. 1.

Базовый образец

В качестве *базового образца* [4, 11] использовалась совокупность максимальных значений показателей качества ЭСТ, которых предполагается добиться путем совершенствования его состава и структуры.

Уровень качества определялся путем сравнения показателей оцениваемого образца с показателями качества не только гипотетического изделия, аналогичного оцениваемому и принимаемому в качестве базового образца, но и ряда существующих изделий. Такой подход важен, так как свойства подобных изделий ока-

Таблица 1

Система показателей качества для оценки научно-технического уровня электронных словарей и энциклопедий

№ пп	Название	Характеризуемое свойство	Метод: Шкала	Вес
1.	**Обеспечение**	**Характеризует совокупность методов, средств и мероприятий, необходимых для нормального функционирования автоматизированной системы (только в части информационной базы электронных словарей и энциклопедий)**	**Расчетный: 0,00...1,0**	**1**
1.1.	Информационное	Характеризует реализованные решения по размещению и формам существования информации в информационной базе электронных словарей и энциклопедий	Расчетный: 0,00...1,00	0,6
1.2.	Программное	Характеризует реализованные решения по совокупности программ на носителях данных, предназначенных для функционирования информационной базы электронных словарей и энциклопедий	Расчетный: 0,00...1,00	0,4
1.1	**Информационное**			
1.1.1.	Структурность	Характеризует наличие в информационной базе определенных элементов, объединяемых в единое целое при представлении информации	Расчетный: 0,00...1,00	0,5
1.1.2.	Организованность	Характеризует наличие некоторой внутренней упорядоченной совокупности, согласующей функциональное взаимодействие элементов информационной базы	Расчетный: 0,00...1,00	0,4
1.1.3.	Развернутость	Характеризует наличие определенного вида дополнительных сведений, выходящих за рамки основного назначения словаря (энциклопедии)	Расчетный: 0,00...1,00	0,1
1.1.1.	**Структурность**			
1.1.1.1.	Многокомпонентность	Наличие основных компонентов словарной статьи (термин, краткая форма, аббревиатура, происхождение, определение, др.)	Регистрационный: 1...6	0,2
1.1.1.2.	Многоабзацность	Построение разъяснения понятия в виде текста (а не только наличие определения в виде одного абзаца)	Регистрационный: 0/1	0,15
1.1.1.3.	Иллюстративность	Применение рисунков, схем, картинок и другого иллюстративного материала для наглядного раскрытия понятий	Регистрационный: 0/1	0,15
1.1.1.4.	Наличие таблиц	Применение таблиц для наглядного представления свойств понятия	Регистрационный: 0/1	0,15
1.1.1.5.	Наличие формул	Приведение формул сложной структуры для описания математического аппарата понятия	Регистрационный: 0/1	0,05
1.1.1.6.	Связанность	Наличие связей между понятиями (объектами) словаря (энциклопедии) в виде гиперссылок	Регистрационный: 0/1	0,2
1.1.1.7.	Ссылочность	Указание ссылок на первоисточники (литературу), откуда взят представляемый материал	Регистрационный: 0 — нет. 0,5 — частично. 1 — есть	0,1
1.1.2.	**Организованность**			
1.1.2.1.	Иерархичность	Представление понятий (объектов) словаря (энциклопедии) в виде иерархического дерева	Регистрационный: 0 — нет. 0,5 — по-другому. 1 — есть	0,35
1.1.2.2.	Ранговость	Наличие определенного порядка (а не только по алфавиту) следования элементов дерева	Регистрационный: 0 — нет. 0,5 — частично. 1 — есть	0,15
1.1.2.3.	Последовательность	Представление всей совокупности понятий (объектов) словаря (энциклопедии) в виде единой алфавитной последовательности	Регистрационный: 0/1	0,3
1.1.2.4.	Многоаспектность	Наличие других указателей для поиска понятий (объектов) словаря (энциклопедии) таких как указатели аббревиатур, по главному слову, таблиц, иллюстраций, первоисточников	Регистрационный: 0...5	0,2
1.1.3.	**Развернутость**			
1.1.3.1.	Персональность	Наличие библиографического раздела или статей персоналий в словаре (энциклопедии)	Регистрационный: 0/1	0,3

Окончание таблицы 1

№ пп	Название	Характеризуемое свойство	Метод: Шкала	Вес
1.1.3.2.	Справочность	Наличие раздела или дополнительных сведений по нормативно-справочной информации	Регистрационный: 0/1	0,4
1.1.3.3.	Мульти-медийность	Наличие видеоматериалов и (или) больших подборок фотографий	Регистрационный: 0/1	0,3
1.2.	**Программное**			
1.2.1.	Надежность	Характеризует способность программы выполнять заданные функции в соответствии с программными документами в условиях возникновения отклонений в среде функционирования, вызванных сбоями технических средств, ошибками во входных данных, ошибками обслуживания и другими дестабилизирующими воздействиями	Расчетный: 0,00...1,00	0,1
1.2.2.	Удобство применения	Характеризует свойства программы, способствующие быстрому освоению, применению и эксплуатации программы с минимальными трудозатратами с учетом характера решаемых задач и требований к квалификации обслуживающего персонала	Расчетный: 0,00...1,00	0,2
1.2.3.	Универсальность	Характеризует адаптируемость программы к новым функциональным требованиям, возникающим вследствие изменения области применения, или других условий функционирования	Расчетный: 0,00...1,00	0,5
1.2.4.	Конструк-тивность	Характеризует конструктивные особенности программы, обусловливающие возможность реализации в программе его функциональных свойств	Расчетный: 0,00...1,00	0,2
1.2.1.	**Надежность**			
1.2.1.1.	Работоспо-собность	Способность программы функционировать в заданных режимах и объемах обрабатываемой информации при отсутствии сбоев технических средств (под ОС Windows 7)	Регистрационный: 0 — не работает. 0,5 — ошибки. 1 — работает	1
1.2.2.	**Удобство применения**			
1.2.2.1.	Легкость освоения	Представление программы в виде, способствующем пониманию логики функционирования программы в целом и ее частей, а также удобство представления сведений (в том числе эргономичность)	Экспертный: по пятибальной шкале	0,5
1.2.2.2.	Тиражируемость	Возможность работы не только с оригинальным машинным носителем	Регистрационный: 0/1	0,5
1.2.3.	**Универсальность**			
1.2.3.1.	Мобильность	Возможность использования программы без предварительной установки на компьютер и регистрации в среде операционной системы (запуск со сменного машинного носителя информации и установка на компьютер простым копированием)	Регистрационный: 0/1	0,3
1.2.3.2.	Открытость	Возможность доступа к ресурсам электронного издания средствами операционной системы и общего программного обеспечения	Регистрационный: 0 — свой формат 0,5 — частично. 1 — типовой.	0,7
1.2.4.	**Конструктивность**			
1.2.4.1.	Многотомность	Возможность работы с разными электронными изданиями (книгами, словарями, томами) из программной среды	Регистрационный: 0 — нет. 0,5 — переключение. 1 — одновременно	0,3
1.2.4.2.	Функцио-нальность	Наличие типовых функций работы со словарями (энциклопедиями) (поиск, закладки, заметки, печать, настройки)	Регистрационный: 0...5	0,7
1.2.4.3.	Общность	Отсутствие специальной программы для работы с информационной базой	Регистрационный: 0 — присутствует. 0,5 — как с программой, так и без нее. 1 — отсутствует	0,0

зывают большое влияние не только на разработку *базовых значений показателей качества продукции* [3], но и на создание всей системы показателей качества.

По причине, изложенной выше, разработка базовых значений показателей качества ЭСТ проводилась с учетом свойств аналогичных изделий. В качестве таких изделий использованы только *электронные издания*, представляющие собой электронные документы (группы электронных документов), прошедшие редакционно-издательскую обработку, предназначенные для распространения в неизменном виде и имеющие выходные сведения [12]. В связи с этим обстоятельством для оценки качества электронных словарей и энциклопедий было использовано 54 электронных словаря и энциклопедии, разработанных в течение последних 10—15 лет, выпущенных на машинных носителях, которые можно приобрести в магазинах или посмотреть в библиотеках. Было решено отказаться от оценки изданий, представленных в сети Интернет, так как на момент обращения читателя к соответствующему сайту он уже может не существовать.

Методы определения значений показателей качества

Для определения показателей качества электронных словарей и энциклопедий использованы расчетный, регистрационный и экспертный методы. Для оценки значений показателей нижнего уровня (оценочных элементов) в зависимости от особенностей характеризуемых ими свойств использованы порядковые (ранговые) и номинальные (классификационные) шкалы.

Основным методом определения оценочных элементов (см. табл. 1) являлся регистрационный метод, фиксирующий значение по принципу 0/1 (нет/есть). В ряде случаев использовалась шкала с промежуточным значением, а также фиксирующая наличие некоторых типовых элементов (функций).

В соответствии со вторым принципом квалиметрии при расчете комплексных показателей использовались не абсолютные значения показателей нижнего уровня, а относительные, определяемые следующим образом:

$$K_{ij}^{l} = \frac{P_{ij}}{\max(P_{ij}, P_{ij\ \text{баз}})},$$

где P_{ij} — абсолютное значение показателя оцениваемого образца; $P_{ij\ \text{баз}}$ — базовое значение показателя; l — нижний уровень иерархии системы показателей качества ($l = 4$); i — номер показателя вышестоящего уровня; j — номер оцениваемого показателя.

В соответствии с пятым принципом квалиметрии каждый показатель характеризуется весовым коэффициентом (см. табл. 1). Такие коэффициенты выбирались из соображений степени важности каждого показателя. Сумма весовых коэффициентов C_{ij}^{l+1} показателей каждого уровня, относящихся к одному показателю предыдущего уровня, является величиной постоянной, которая принимается равной единице:

$$\sum_{j=1,\,n} C_{ij}^{l+1} = 1,$$

где n — число показателей нижестоящего уровня $(l + 1)$, относящихся к i-му показателю уровня l.

В соответствии с первым принципом квалиметрии, оценка качества проводилась по уровням иерархии системы, начиная с нижнего. Для относительных значений показателей на всех уровнях иерархической структуры принята единичная шкала от 0 до 1. Показатели качества каждого вышестоящего уровня определялись показателями нижестоящего. Показатели качества вышестоящих уровней (в том числе и научно-технического уровня) определялись суммированием подчиненных им показателей с учетом их весовых коэффициентов:

$$K_{ij}^{l} = \sum_{j=1}^{n} K_{ij}^{l+1} C_{ij}^{l+1},$$

где $l = 0...3$ — уровни иерархии, содержащие комплексные показатели.

Принимая во внимание изложенные выше соображения, с учетом разработанной системы показателей качества, комплексные показатели рассчитывались в соответствии с формулами, представленными в табл. 2.

Таблица 2

Формулы расчета комплексных показателей качества

№ пп	Формула расчета
1.	$K_{11}^{1} = K_{11}^{2}0{,}6 + K_{12}^{2}0{,}4$
1.1	$K_{11}^{2} = K_{11}^{3}0{,}5 + K_{12}^{3}0{,}4 + K_{13}^{3}0{,}1$
1.1.1.	$K_{11}^{3} = \frac{K_{11}^{4}}{6}0{,}2 + K_{12}^{4}0{,}15 + K_{13}^{4}0{,}15 + K_{14}^{4}0{,}15 + K_{15}^{4}0{,}05 + + K_{16}^{4}0{,}2 + K_{17}^{4}0{,}1$
1.1.2	$K_{12}^{3} = K_{21}^{4}0{,}35 + K_{22}^{4}0{,}15 + K_{23}^{4}0{,}3 + \frac{K_{24}^{4}}{5}0{,}2$
1.1.3	$K_{13}^{3} = K_{31}^{4}0{,}3 + K_{32}^{4}0{,}4 + K_{33}^{4}0{,}3$
1.2.	$K_{12}^{2} = K_{21}^{3}0{,}1 + K_{22}^{3}0{,}2 + K_{23}^{3}0{,}5 + K_{24}^{3}0{,}2$
1.2.1.	$K_{21}^{3} = K_{11}^{4}1{,}0$
1.2.2.	$K_{22}^{3} = \frac{K_{21}^{4}}{5}0{,}5 + K_{22}^{4}0{,}5$
1.2.3.	$K_{23}^{3} = K_{31}^{4}0{,}3 + K_{32}^{4}0{,}7$
1.2.4.	$K_{24}^{3} = K_{41}^{4}0{,}3 + \frac{K_{42}^{4}}{5}0{,}7 + K_{43}^{4}0{,}0$

Определение значений показателей качества

Определение значений показателей качества электронных словарей и энциклопедий осуществлялось для отобранной совокупности изданий по выбранной номенклатуре таких показателей с помощью определенных методов вычисления их значений. В табл. 3 в качестве примера представлены значения показателей качества, полученные для следующих пяти наиболее интересных и характерных изданий:

- Электронный словарь терминов, создаваемый авторским коллективом;
- Военная Россия: Авиация. М.: МедиаХауз; Русский военно-исторический фонд, 2004. 1 электрон. опт. диск (CD-ROM);
- Энциклопедия современных вооружений (версия 2.0). М.: Акелла, 2002. 1 электрон. опт. диск (CD-ROM);
- Лукашевич В. П. Мультимедийная энциклопедия "Буран" (версия 3.20, выпуск от 10.08.2002). URL: http://www.buran.ru (дата обращения: 13.12.2011). 3 электрон. опт. диска (CD-ROM);
- Военная энциклопедия (т-во И. Д. Сытина, 1911—1915 гг.). М.: ИДДК, 2007. 4 электрон. опт. диска (CD-ROM).

Для того чтобы проанализировать общие тенденции уровня качества, проведено сведéние большого количества показателей к одному или нескольким, которые характеризуют основные зависимости проводимого исследования. Отметим, что при таком подходе даже в случае потери точности полученных значений поставленная цель оценки достигается.

Реализуя такой подход к оценке, вся совокупность рассчитанных комплексных показателей нижнего уровня была обобщена по каждой компании (группе компаний, организации, сайту), осуществившей публикацию, путем выбора лучших значений:

$$K_{ij}^{l} = \max_{m = 1,\, n} K_{ij}^{lm},$$

где l — нижний уровень иерархии системы показателей качества ($l = 4$); n — число изданий, опубликованных компанией, группой компаний, организацией или сайтом; K_{ij}^{lm} — показатель качества K_{ij}^{l} каждого издания.

Комплексные показатели более высоких уровней были пересчитаны со снижением точности до одного знака после запятой.

Полученные значения научно-технического уровня оцениваемых изданий по каждой компании, которые характеризуются по выбранной номенклатуре показателей качества только одним комплексным показателем верхнего уровня "Обеспечение", представлены в табл. 4. Для удобства восприятия представленные результаты отсортированы по убыванию итогового показателя.

Такое представление данных позволяет отобрать компании, имеющие издания с более высоким уров-

Таблица 3

Пример определения значений показателей качества электронных словарей и энциклопедий

Оцениваемые изделия	1.	1.1.	1.1.1.	1.1.1.1.	1.1.1.2.	1.1.1.3.	1.1.1.4.	1.1.1.5.	1.1.1.6.	1.1.1.7.	1.1.2.	1.1.2.1.	1.1.2.2.	1.1.2.3.	1.1.2.4.	1.1.3
Электронный словарь терминов	0,84	0,78	0,80	6	1	1	0	0	1	1	0,96	1	1	1	4	0,00
Военная Россия: Авиация	0,76	0,72	0,73	2	1	1	1	0	1	0,5	0,64	0,5	1	0,5	4	1,00
Энциклопедия вооружений	0,76	0,84	0,85	3	1	1	1	0	1	1	0,79	0,5	1	1	4	1,00
Энциклопедия "Буран"	0,70	0,62	0,78	1	1	1	1	0	1	1	0,33	0,5	1	0	0	1,00
Военная энциклопедия Сытина	0,49	0,50	0,52	2	1	1	1	0	0	0	0,46	0	0	1	4	0,60

Продолжение табл. 3

Оцениваемые изделия	1.1.3.1.	1.1.3.2.	1.1.3.3.	1.2.	1.2.1.	1.2.1.1.	1.2.2.	1.2.2.1.	1.2.2.2.	1.2.3.	1.2.3.1.	1.2.3.2.	1.2.4.	1.2.4.1.	1.2.4.2.	1.2.4.3.
Электронный словарь терминов	0	0	0	0,92	1,00	1	1,00	5	1	1,00	1	1	0,58	1	2	1
Военная Россия: Авиация	1	1	1	0,83	1,00	1	1,00	5	1	1,00	1	1	0,14	0	1	1
Энциклопедия вооружений	1	1	1	0,64	0,00	0	0,90	4	1	0,70	0	1	0,56	0	4	0
Энциклопедия "Буран"	1	1	1	0,80	1,00	1	0,90	4	1	1,00	1	1	0,14	0	1	0,5
Военная энциклопедия Сытина	1	0	1	0,47	0,50	0,5	0,50	5	0	0,35	0	0,5	0,71	0,5	4	0

Таблица. 4

Значения комплексных показателей оценки электронных словарей и энциклопедий, сгруппированные по компаниям, осуществившим их публикацию

Издатель (число изданий)	**1**	**1.1**	1.1.1	1.1.2	1.1.3	**1.2**	1.2.1	1.2.2.	1.2.3	1.2.4
Директмедиа Паблишинг (2)	**0,9**	**0,8**	0,6	1,0	0,7	**1,0**	1,0	0,9	1,0	1,0
Авторы	**0,8**	**0,8**	0,8	1,0	0,0	**0,9**	1,0	1,0	1,0	0,6
Акелла (2)	**0,8**	**0,8**	0,9	0,8	1,0	**0,6**	0,0	0,9	0,7	0,6
МедиаХауз (23)	**0,8**	**0,7**	0,7	0,6	1,0	**0,8**	1,0	1,0	1,0	0,1
КогАх (4)	**0,7**	**0,5**	0,5	0,5	0,7	**0,9**	1,0	1,0	1,0	0,3
ship.bsu.by (1)	**0,7**	**0,7**	0,8	0,5	0,7	**0,8**	1,0	1,0	1,0	0,0
www.buran.ru (1)	**0,7**	**0,6**	0,8	0,3	1,0	**0,8**	1,0	0,9	1,0	0,1
BornToKill (1)	**0,6**	**0,4**	0,4	0,5	0,0	**0,8**	1,0	0,9	1,0	0,6
Mylimedia (1)	**0,6**	**0,5**	0,5	0,4	0,6	**0,9**	1,0	0,8	1,0	0,5
SoftBeep (1)	**0,6**	**0,5**	0,5	0,5	0,4	**0,9**	1,0	1,0	1,0	0,3
ИДДК (3)	**0,6**	**0,5**	0,5	0,5	0,6	**0,8**	1,0	1,0	0,7	0,7
МастерМедиа (1)	**0,6**	**0,4**	0,3	0,5	0,7	**0,8**	0,0	1,0	1,0	0,4
Медиа-Сервис 2000 (3)	**0,6**	**0,5**	0,5	0,3	1,0	**0,8**	1,0	0,9	1,0	0,3
Руссобит-М (1)	**0,6**	**0,6**	0,5	0,8	0,3	**0,6**	1,0	1,0	0,3	0,6
Самиздат (2)	**0,6**	**0,4**	0,4	0,4	0,0	**0,9**	1,0	1,0	1,0	0,4
ТОВ "Мультитрейд" (2)	**0,6**	**0,4**	0,5	0,5	0,0	**0,9**	1,0	1,0	1,0	0,6
flot.sevactopol.info (1)	**0,5**	**0,5**	0,5	0,3	1,0	**0,5**	1,0	1,0	0,3	0,0
ИД "Равновесие" (1)	**0,5**	**0,5**	0,4	0,6	0,7	**0,6**	1,0	1,0	0,4	0,4
Новый диск (1)	**0,5**	**0,3**	0,3	0,3	0,0	**0,7**	1,0	0,9	0,7	0,4
Russian-ship.info (1)	**0,4**	**0,3**	0,4	0,2	0,4	**0,5**	1,0	1,0	0,3	0,0
Музей им. Маринеско (1)	**0,4**	**0,3**	0,3	0,2	0,6	**0,7**	1,0	0,6	1,0	0,0
Сталкер Медиа (1)	**0,4**	**0,3**	0,3	0,2	0,3	**0,6**	1,0	0,9	0,7	0,0

нем качества, для дальнейшего изучения предложенных в настоящей публикации решений и использования заложенных в них идей и методов, направленных на совершенствование ЭСТ.

Оценка научно-технического уровня

Для упорядочения и упрощения процедуры сравнения для каждого конкретного случая оценки научно-технического уровня одного изделия заранее устанавливаются используемые шкалы и способы их приведения к сопоставимым значениям [13].

В связи с тем, что при оценке качества электронных словарей и энциклопедий использовались различные шкалы оценки показателей (шкалы базовых значений показателей качества, единая или типовая порядковая шкала, произвольные порядковые шкалы) для оценки научно-технического уровня K^0 выбрана наиболее простая и понятная шкала. Данная шкала, основанная на пятибальной системе, представлена в табл. 5.

С помощью этой шкалы также могут быть оценены все другие комплексные и единичные показатели.

Результаты оценки, представленные в табл. 4, показали отсутствие изданий с неудовлетворительным

Таблица 5

Шкала для оценки научно-технического уровня

Оценка	Значение научно-технического уровня
Отлично	$0,8 \leqslant K^0 \leqslant 1,0$
Хорошо	$0,6 \leqslant K^0 \leqslant 0,8$
Удовлетворительно	$0,4 \leqslant K^0 \leqslant 0,6$
Неудовлетворительно	$K^0 < 0,4$

качеством. Издания первой тройки компаний и авторская разработка получили отличную оценку научно-технического уровня, издания большинства остальных компаний — хорошую. Отличный научно-технический уровень разрабатываемого электронного словаря терминов объясняется не только реализуемыми решениями, принятыми при его построении, но и ориентацией выбранной номенклатуры показателей качества на его состав и структуру.

Подводя итог проведенного исследования необходимо заметить, что оценка электронных изданий по данной методике во многом зависит от системы показателей качества, установленной в конкретной организации или принятой при проведении научно-исследовательской или опытно-конструкторской работы. При смене номенклатуры показателей качества или способов расчета абсолютных показателей результаты оценок могут существенно измениться.

Заключение

Оценка показателей качества или технического (научно-технического) уровня продукции как товарной, так и информационной, может осуществляться различными методами и способами. Одним из них является методологический аппарат квалиметрии. Его применение позволяет количественно оценить качество объектов любой природы, таких как материя, информация, энергия, сознание, процесс, явление или услуга. Суть аппарата этой науки состоит в получении единичных и комплексных количественных показателей, характеризующих уровень оцениваемых изделий (объектов, процессов).

Представленная в статье методика оценки качества, основанная на принципах квалиметрии, позволяет: оценить качество изделий, используя различные системы показателей качества; закладывать качество изделий при разработке технического задания и контролировать его на всех этапах жизненного цикла, т. е. оценивать минимальный уровень качества при неполной информации об изделии, который достигнут при уже полученных значениях показателей качества; основываясь на установленной системе показателей качества, проводить оценку разных изделий одинакового назначения в целях выявления лучшего их них.

Целью оценки качества электронных словарей и энциклопедий являлось выявление таких их качеств, реализация которых в электронном словаре терминов позволила бы достичь лучшего уровня качества электронного словаря в соответствии с его предназначением в автоматизированной системе. Электронные словари и энциклопедии в автоматизированной системе являются компонентом ее лингвистического обеспечения и могут разрабатываться как информационные или программные изделия в автоматизированной системе. Выбранная система показателей качества рассматривает электронные словари и энциклопедии как составную часть информационного и программного обеспечения автоматизированной системы. Это обстоятельство позволяет встраивать ее в полные системы оценки показателей качества автоматизированных систем, которые разрабатываются на основе требований технического задания. Встраивание при этом осуществляется как на уровне общесистемных требований, так и на уровне требований к видам обеспечения автоматизированной системы.

При разработке показателей, оценивающих информационное обеспечение, за основу были взяты особенности построения базы данных электронного словаря терминов и его информационного наполнения. Это позволило определить не только достигнутый уровень его разработки, но и выявить направления его развития. Отличная оценка разрабатываемого электронного словаря терминов может быть объяснена не только реализованными решениями, принятыми при его построении, но и ориентацией выбранной номенклатуры показателей качества на его состав и структуру.

Список литературы

1. **РД 50-680—88.** Методические указания. Автоматизированные системы. Основные положения. М.: Изд-во стандартов, 1991. 7 с.

2. **ГОСТ 34.003—90.** Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения. М.: Изд-во стандартов, 1991. 23 с.

3. **ГОСТ 15467—79.** Управление качеством продукции. Основные понятия. Термины и определения. М.: Изд-во стандартов, 1981. 26 с.

4. **Кириллов В. И.** Квалиметрия и системный анализ. М.: ИНФРА-М, 2011. 440 с.

5. **Лобанов А. С.** Управление качеством. М.: МАКС Пресс, 2009. 292 с.

6. **ГОСТ 16504—81.** Система государственных испытаний продукции. Испытания и контроль качества продукции. Основные термины и определения. М.: Изд-во стандартов, 1985. 28 с.

7. **Гличев А. В., Панов В. П., Азгальдов Г. Г.** Что такое качество? М.: Экономика, 1968. 135 с.

8. **Азгальдов Г. Г., Зорин В. А., Павлов А. П.** Квалиметрия для инженеров-механиков. М.: МАДИ (ГТУ), 2006. 145 с.

9. **Азгальдов Г. Г., Райхман Э. П.** О квалиметрии / Под ред. А. В. Гличева. М.: Изд-во стандартов, 1973. 172 с.

10. **ГОСТ 7.0—99.** Система стандартов по информации, библиотечному и издательскому делу. Информационно-библиотечная деятельность, библиография. Термины и определения. Минск: ИПК Изд-во стандартов, 1999. 23 с.

11. **Фомин В. Н.** Квалиметрия. Управление качеством. Сертификация. М.: Ось-89, 2007. 384 с.

12. **ГОСТ 7.83—2001.** Система стандартов по информационному библиотечному и издательскому делу. Электронные издания. Основные виды и выходные сведения. Минск: ИПК Изд-во стандартов, 2002. 13 с.

13. **Кулаков А. Ф.** Управление качеством программных средств ЭВМ. Киев: Тэхника, 1989. 216 с.

УДК 004.9

А. А. Попов, д-р физ.-мат. наук, проф.,
Марийский государственный университет,
e-mail: apopov@vvoi.ru

Методика программирования на языке Java тренажеров по математике с посимвольным контролем аналитических преобразований

Предложен подход, в котором аналитическое решение дифференциального уравнения представлено в виде массива строк языка Java со специально закодированными последовательностями символов. Каждому символу строки поставлено в соответствие его графическое представление, включающее значение символа, его координаты, размеры, цвет и признаки, определяющие принадлежность символа числителю или знаменателю дроби. Преобразованные данные для каждого символа собраны в массивы, которые являются основой для иллюстрирующих программ, обучающих программ и программ-тренажеров. Рассмотрены особенности программирования и приведены примеры реализаций линейных дифференциальных уравнений при их решениях методами Лагранжа и Бернулли-Фурье.

Ключевые слова: *программирование, язык Java, эталонное решение, обучающие программы, тренажеры, дифференциальные уравнения, метод Лагранжа, метод Бернулли-Фурье*

Введение

Преподавание таких математических дисциплин как высшая математика, математический анализ, дифференциальные уравнения должно не только гарантировать предоставление знаний по дисциплине, но и обеспечивать выработку навыков решения задач различной сложности. С развитием компьютерных технологий появилась возможность нетворческую работу, связанную с контролем решения задач, возложить на компьютер. Существуют различные варианты контроля, но в целом их можно свести к двум видам: контроль правильности окончательных формул и контроль процесса получения решения, в том числе и окончательных формул. Первый вид контроля определяет тестирующие системы, второй вид соответствует тренажерам.

В настоящее время хорошо развит только первый вид контроля. Наиболее типичные и проработанные программные продукты представлены в работах [1—3]. В основе каждого из них лежит язык разметки математических текстов MathML. Данный язык используется для ввода математических формул, представляющих решение задачи, а различное кодирование символов, цифр и знаков операций предоставляет возможность или проводить вычисления введенных выражений, или находить позиции контролируемых в выражении коэффициентов. В статьях [1, 2] представлены, исходя из названий, тренажеры, однако они больше похожи на сложные тестирующие системы. В них правильность введенного выражения контролируется путем разделения его на части с последующим вычислением этих частей в заданных точках. Понятно, что начать такой контроль можно только в том случае, когда выражение полностью введено в программу. На сайте [3] о тренажере свидетельствует только его название, поскольку в меню о тренажере не упоминается, а речь идет о лекциях, задачах и тестах. Причем в тестах заданы шаблоны математических выражений, в которые необходимо вставить коэффициенты.

В работах автора [4, 5] предложен другой подход, который позволяет проводить контроль процесса решения задачи после каждого введенного символа. Для осуществления данного вида контроля необходимо предварительно "научить" компьютер решать математические задачи, что возможно только при использовании языка программирования высокого уровня. В работах [4, 5] в качестве такого языка выбран язык Java, в рамках которого один и тот же алгоритм можно запрограммировать и в виде графического приложения, и в виде апплета. На этом языке реализован алгоритм получения эталонного решения задачи в виде массива строк, которым поставлены в соответствие их графические представления. Значение каждого символа вместе с его размерами, цветом и принадлежностью числителю или знаменателю дроби фиксируется в массивах. При таком представлении решения возникает возможность проконтролировать каждый введенный символ, находя ему аналог в эталонном решении. На языке высокого уровня легко организовать и процесс ввода символов, если использовать соответствующий обработчик событий.

В настоящей статье рассматривается методика программирования тренажера на языке Java, в котором контролируется процесс получения аналитического решения математической задачи. В качестве примера выбрано решение линейного неоднородного дифференциального уравнения первого порядка с помощью классических методов Лагранжа и Бернулли-Фурье.

Формулы для программирования

Линейное неоднородное дифференциальное уравнение первого порядка имеет вид [6]:

$$y' + P(x)y = Q(x).$$

Для его решения существуют два метода: метод Лагранжа и метод Бернулли-Фурье. Оба метода сводят исходное уравнение к двум дифференциальным уравнениям с разделяющимися переменными. При интегрировании дифференциального уравнения методом Лагранжа решение может быть представлено следующими формулами [6]:

$$y = C(x)\mathbf{e}^{-\int P(x)dx},$$

$$C(x) = \int Q(x)\mathbf{e}^{\int P(x)dx}dx + C_0.$$

При использовании метода Бернулли-Фурье решение отыскивается в виде произведения двух функций $y = uv$, каждая из которых получается путем интегрирования соответствующего дифференциального уравнения с разделяющимися переменными и имеет вид [6]:

$$v = \mathbf{e}^{-\int P(x)dx},$$

$$u = \int Q(x)\mathbf{e}^{\int P(x)dx}dx + C.$$

Общее решение $y = uv$ совпадает с решением, полученным методом Лагранжа, но алгоритмы этих методов значительно отличаются.

Введем в рассмотрение новые функции, через которые можно выразить все промежуточные формулы при сведении дифференциального уравнения к общему решению как для метода Лагранжа, так и для метода Бернулли-Фурье. Бóльшую часть линейных дифференциальных уравнений можно отнести к одной из двух групп. Для первой группы задач необходимые функции $a(x)$ и $b(x)$ вводятся следующим образом:

$$\begin{aligned} a(x) &= \int P(x)dx \Rightarrow P(x) = a'(x), \\ b(x) &= \int Q(x)\mathbf{e}^{a(x)}dx \Rightarrow Q(x) = b'(x)\mathbf{e}^{-a(x)}. \end{aligned} \quad (1)$$

Для задач данной группы решение будет содержать экспоненциальный сомножитель, который образуется при использовании основного логарифмического тождества в процессе преобразования первообразной функции, полученной после интегрирования уравнения с разделенными переменными. Для задач второй группы рассматриваются функции $g(x)$ и $h(x)$. Функцией $g(x)$ обозначается весь экспоненциальный сомножитель, а функция $h(x)$ связана с $g(x)$:

$$g(x) = \mathbf{e}^{\int P(x)dx} \Rightarrow P(x) = \frac{g'(x)}{g(x)},$$

$$h(x) = \int Q(x)g(x)dx \Rightarrow Q(x) = \frac{h'(x)}{g(x)}.$$

Для задач второй группы правая часть уравнения с разделенными переменными без каких-либо преобразований сводится к логарифму. В итоге окончательное решение не содержит экспоненциальный сомножитель. Через введенные функции, их производные и дифференциалы можно определить весь процесс сведения линейного дифференциального уравнения к общему решению.

Особенности программирования

Функции $a(x)$, $b(x)$, $g(x)$ и $h(x)$ выбираются случайным образом из списка элементарных функций:

$$\cos(...),\ \sin(...),\ \mathrm{tg}(...),\ \mathrm{ctg}(...),$$
$$\arcsin(...),\ \mathrm{arctg}(...),\ (...)^n,\ \sqrt{...}.$$

Кроме того, функции $b(x)$ и $h(x)$ могут быть представлены многочленами. К каждой функции составляются производная, дифференциалы различного вида,

а для $g(x)$ производная логарифма этой функции. Аргументом элементарной функции выбрано выражение вида:

$$A = rx^2 + (1 - r)px + q. \quad (2)$$

Число r может принимать значения 0 или 1, т. е. аргумент является линейной функцией или квадратичной функцией, в которой отсутствует линейное слагаемое. Целые числа p и q генерируются случайным образом.

Каждый метод имеет по два разных варианта для составления так называемого эталонного решения. Это решение включает все аналитические преобразования, сводящие дифференциальное уравнение к общему решению. Оно отыскивается в виде массива строк символов на языке Java [7]. Например, часть одного из вариантов для дифференциального уравнения, решаемого методом Бернулли, имеет вид:

```
s[0] = "y'"+Sign+P+"y = "+Q;
s[1] = "y = uv";
s[2] = "u'v+uv'+Sign+P+"uv = "+Q;
s[3] = "u'v+u(v'"+Sign+P+"v) = "+Q;
s[4] = "v'"+Sign+P+"v = 0";
s[5] = "u'v = "+Q;
s[6] = "<dv>{dx} = "+Pm+"v";
s[7] = "<dv>{v} = "+Pmdx.
```

(3)

Символом s обозначен массив из строк Sign — пустая строка или "+"; P и Q — строки, формируемые из элементарных функций и их производных. Если предположить, что реализован первый вариант решения, а в качестве функции $a(x)$ и $b(x)$ выбраны, например, функции sin(...) и cos(...), то на основе представлений (1) строки P и Q примут вид:

$$P = "\cos("+A1+")"+As1;$$

$$Q = "-\sin("+A2+")"+As2+"\mathbf{e}[-\sin("+A1+")]";$$

где $A1$, $A2$ — строки для аргументов, полученных на основе формулы (2), $As1$ и $As2$ — строки для производных аргументов. Указанные строки формируются программным путем. Остальные строки в коде (3) также формируются автоматически. Строка Pm — строка P с измененным знаком, $Pmdx$ — строка Pm с сомножителем "dx". Если Pm является дробью, то "dx" добавляется в числитель.

В приведенной части кода исходное уравнение разбивается на два дифференциальных уравнения, затем в первом уравнении для функции v разделяются переменные. В зависимости от метода или выбранного варианта решения таких строк может быть от 21 до 25.

Каждая строка заранее кодируется так, чтобы из нее однозначно следовало графическое представление. Например, если в процессе получения эталонного решения образовались строки

$$\ln y = I < d(x[2] + 1) > \\ > \{\mathrm{arctg}(x[2] + 1)(1 + (x[2] + 1)[2])\} + \ln C,$$

$$u'v + u(v' + <2>\{|4x + 6\&\}v) = \\ = -<5>\{\sin[2](5x - 3)\}\mathbf{e}[-|4x + 6\&],$$

то им будут соответствовать следующие графические представления:

$$\ln y = \int \frac{d(x^2 + 1)}{\mathrm{arctg}(x^2 + 1)(1 + (x^2 + 1)^2)} + \ln C,$$

$$u'v + u\left(v' + \frac{2}{\sqrt{4x + 6}}v\right) = -\frac{5}{\sin^2(5x - 3)}\mathbf{e}^{-\sqrt{4x+6}}.$$

Символом I обозначен знак интеграла, символом | знак квадратного корня, а все символы, расположенные между | и &, принадлежат подкоренному выражению. Символы, помещенные в квадратные скобки, располагаются в показателе степени. Квадратные скобки могут иметь вложенные в них другие квадратные скобки. В данном случае показатель степени интерпретируется как степенная функция. Дробь образуется парами угловых и фигурных скобок. Выражения, заключенные в этих скобках, помещаются соответственно в числитель и знаменатель дроби. Средствами класса FontMetrics [7] измеряются длины числителя и знаменателя дроби в пикселях. Эти размеры используются для центрирования выражения с наименьшей длиной, а по наибольшей длине определяется длина дробной черты.

Наиболее привычный вид иллюстрации решения задачи — когда к текущему выражению добавляется по одному символу, а при заполнении окна часть промежуточных формул делается невидимой, освобождая место для следующих символов. Такое представление решения эквивалентно написанию символов на аудиторной доске с последующим освобождением ее части для новых формул. Каждый символ решения должен быть выведен в определенное место окна, т. е. кроме значения символа должны быть заранее известны его координаты и размеры. Данная информация фиксируется в массивах. Также сохраняются в массиве данные о цвете символов и некоторый признак, задающий принадлежность символа числителю или знаменателю. Если при иллюстрации решения последняя информация для символа является избыточной, то для тренажера, который будет контролировать правильность ввода символов, определяющих решения, она является необходимой. Невидимость строк решения обеспечивается значением логической переменной, связанной с каждым символом.

Иллюстрирующие и обучающие программы

Преобразованное в массивы эталонное решение определяет основу для компьютерных приложений различного вида. Однако каждое из приложений, будь то иллюстрирующая программа, обучающая программа или программа-тренажер, требует дальнейшей доработки. Наиболее просто решается задача, связанная с иллюстрацией методов Лагранжа и Бернулли-Фурье на лекциях или практических занятиях. Данные в виде массивов удобно использовать для посимвольного вывода решения. Достаточно реализовать обработку события, связанного, например, с нажатием заранее выбранной клавиши, при котором счетчик числа выведенных символов увеличивался на 1. В программу также нужно добавить несколько условий, в которых при достижении счетчиком определенных значений группа ранее выведенных на экран символов становится невидимой. Данный вариант иллюстрирующей программы является наиболее простым, но заслуживает внимание потому, что в свою очередь является основой обучающей программы.

Другой вариант иллюстрирующей программы использует интерфейс Runnable, с помощью которого при нажатии клавиши Enter запускается процесс посимвольного вывода эталонного решения. После добавления на экран каждого очередного символа происходит некоторая временная задержка. Не завершая процесс, время задержки можно увеличивать или уменьшать. Существует также возможность останавливать процесс и вновь его возобновлять. Процесс автоматически завершается при выводе на экран последнего символа аналитического решения задачи. Программы подобного вида удобно использовать при иллюстрациях на лекциях, учитывая, что до запуска процесса вывода символов можно щелчком мыши генерировать разные исходные дифференциальные уравнения.

Связать вывод очередного символа с нажатием, например, клавиши Enter, удобно для обучающей программы. В программах этого вида процесс решения сопровождается сообщениями с информацией о том, что в данный момент происходит в решении.

На рис. 1 (см. третью сторону обложки) приведено окно обучающей программы для метода Бернулли-Фурье. Сообщение сопровождает последние выводимые символы, т. е. перевод функции $(-\cos(5x - 3))$ в показатель степени, после которого в правой части выражения образуется логарифм. Первая строка, в которой расположено исходное дифференциальное уравнение, остается неизменной. Остальные строки, соответствующие промежуточным формулам, могут появляться и исчезать в процессе решения. В итоге полное описание иллюстрируемого метода с помощью формул и комментариев разворачивается в пределах одного окна, не содержащего полос прокрутки. Соответствие комментариев нужным формулам обеспечивается оператором множественного выбора *switch*. Комментарий может быть составным, т. е. к нему в процессе вывода символов могут добавляться текстовые строки. Преобразование подынтегрального выражения путем внесения функций под знак дифференциала сопровождается выводом поясняющих формул, которые генерируются вместе с эталонным решением.

На рис. 2 (см. третью сторону обложки) приведен фрагмент решения линейного неоднородного дифференциального уравнения методом Лагранжа. Данный метод замечателен тем, что в нем присутствует проверка правильности формулы решения для однородного дифференциального уравнения. Если эта формула верна, то правильно будет представлен и вид решения неоднородного уравнения. В результате подстановки решения, определенного во второй строке, в исходное уравнение, два слагаемых в полученном дифференциальном уравнении для функции $C(x)$ взаимно уничтожаются (рис. 2, см. третью сторону обложки). Алгоритмы методов Лагранжа и Бернулли-Фурье описываются в комментариях в полном объеме. Программы данного вида предназначены для самостоятельной работы студентов.

Программы-тренажеры

Для данного вида программ эталонное решение используется для контроля вводимых символов. На рис. 3 (см. третью сторону обложки) представлено окно, которое появляется при загрузке тренажера по методу Лагранжа. Такое же окно с первоначальными инструкциями выводится в тренажере по методу Бернулли-Фурье. Комментарий можно убрать, нажав клавишу Delete, или вывести вновь, наведя указатель мыши на прямоугольную область с надписью "Информация". В правом верхнем углу выведены прямоугольники с цифрами 1 и 2. Выделенная цифра 2 соответствует задаче, в которой промежуточное решение для однородного уравнения получается без экспоненциального сомножителя. При щелчке мыши по исходному уравнению генерируется новое дифференциальное уравнение, из которого следует одно из двух вариантов решения. При щелчке по цифрам 1 или 2 генерируется уравнение, принадлежащее выбранному варианту.

Решение задачи вводится с помощью клавиатуры, что соответствует процессу написания символов в тетради. При вводе символов осуществляется многовариантный контроль. Все ошибки связаны с двумя группами символов: лишними и недостающими. Контроль проводится в пределах текущего блока, соответствующего строке решения. Если вводимый символ есть в блоке, то он должен быть введен в необходимую часть выражения: числитель, знаменатель или вне дроби. Не должен быть также нарушен порядок следования символов. Например, если при вводе sin пропущена буква i, то это не является ошибкой. Для вставки i в sn необходимо курсор установить перед n, в противном случае при вводе i программа выдаст ошибку. Все введенные символы, не принадлежащие текущему блоку, являются лишними, и их число фиксируется на месте первого нуля в правом нижнем углу. Переход к следующему блоку осуществляется автоматически. Однако пропущенные символы могут не дать это сделать.

К следующему блоку пользователь может перейти досрочно, нажав клавишу Page Down. Программа вставит все недостающие символы, выделит их красным цветом. Число недостающих символов фиксируется на месте второго нуля в правом нижнем углу.

Программа допускает возможность случайных ошибок, примерно одна ошибка на 80 введенных символов. Заранее известно о числе ошибок, которые можно совершить при вводе символов решения дифференциального уравнения (рис. 3, см. третью сторону обложки). Если число ошибок превысит установленный предел, то пользователь может вернуться к началу решения той же самой задачи после нажатия клавиши Home. Повтор решения одной и той же задачи закрепляет практические навыки при использовании в данном случае методов Лагранжа или Бернулли-Фурье.

Для исключения случайных ошибок курсоры меняют свой вид (рис. 3, см. третью сторону обложки). Например, если символы вводятся под знак радикала, то рядом с курсором расположена буква R. Если курсор выводится из-под знака радикала, то буква R исчезает. Графическое изображение курсора соответствует положению буквы E в исходной символьной строке. Например, при положении E в строке "|1 + 2*x*E&" курсор имеет рядом букву R, так как расположен между символами | и &. В данном случае символы вводятся под радикал. Первоначально строка текущего блока состоит из одного символа E. При нажатии клавиши ↑ курсор переводится в числитель, что автоматически образует строку <E>{}, которая является шаблоном дроби. После ввода в числитель, например, символа *s* (при условии, если он есть в числителе), образуется строка <*s*E>{}, которой соответствует графическое изображение дроби с числителем *s* и дробной чертой с длиной, равной длине строки *s*. После символа *s* будет расположен курсор в виде стрелки, направленной вниз (рис. 3, см. третью сторону обложки).

Размеры символов фиксируются в массивах в виде целых чисел, каждое из которых с помощью оператора *switch* связывается с тем или иным объектом класса *Font*, определяющего шрифт. Если для символа номер шрифта соответствует мелким символам, то к строке этот символ добавляется вместе с квадратными скобками. При выводе данной строки в графическом виде символ в квадратных скобках будет представлен в показателе степени.

На рис. 4 приведено текущее положение окна тренажера по методу Бернулли-Фурье, когда в последней строке окна получено уравнение для функции *v*. Далее на место курсора необходимо переписать дифференциальное уравнение из предпоследней строки с учетом равенства нулю выражения в скобках левой части уравнения. Дифференциальное уравнение для функции *u* временно будет находиться во второй строке, пока не будет получено решение для функции *v*. В отличие от метода Лагранжа в методе Бернулли-Фурье решение исходного дифференциального уравнения сразу разбивается на два уравнения. Для этого необходимо преобразовать левую часть уравнения, а правая часть при этом остается неизменной. В тренажере после ввода всех символов левой части и знака равенства правая часть уравнения переписывается автоматически. После нахождения функции *v* и подстановки ее в уравнение для функции *u*, формула для *v* переписывается тренажером автоматически во вторую строку. Тренажер избавляет пользователя от рутинной работы, связанной с переписыванием одной и той же группы символов, и от случайных ошибок, которые могли бы возникнуть при переписывании формул.

На рис. 5 представлено окно тренажера по методу Лагранжа для решения линейного неоднородного дифференциального уравнения. Оно расположено в первой строке. Со второй по пятую строку располагаются промежуточные формулы. В частности, приведены преобразования, в результате которых получено решение линейного однородного уравнения. В пятой строке не хватает закрывающейся скобки. Если скобка будет введена, то строки со второй по четвертую будут сделаны невидимыми. Курсор автоматически будет переведен в начало второй строки. На его место нужно будет ввести форму общего решения, совпадающего

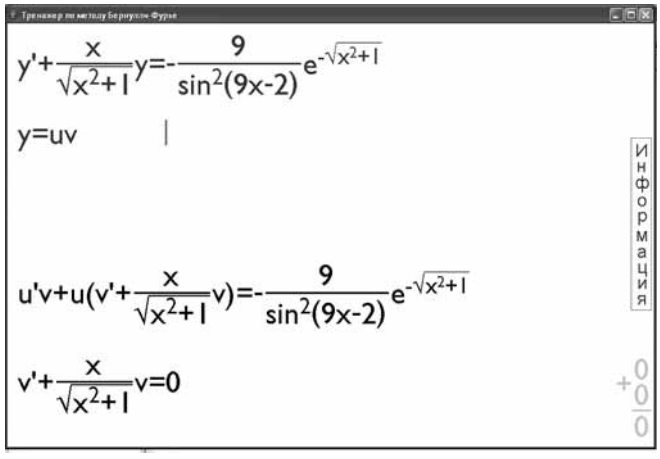


Рис. 4. Окно тренажера по решению линейного дифференциального уравнения методом Бернулли-Фурье

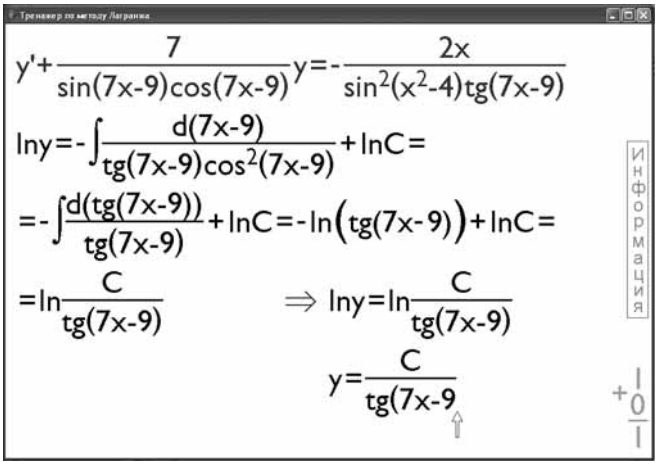


Рис. 5. Окно тренажера по решению линейного дифференциального уравнения методом Лагранжа

с формулой пятой строки с заменой константы C функцией $C(x)$.

При программировании тренажеров необходимо учесть, что существует несколько правильных решений, отличающихся по форме записи. Для безошибочного ввода правильного решения, отличающегося от эталонного, запрограммированы исключения. Например, дифференциал dx в интеграле может быть введен как в числитель, так и вне дроби. При этом тренажер не выдаст ошибку, но расположит dx так, как он определен в эталонном решении. Тренажер по методу Лагранжа имеет больше исключений, поскольку в рамках этого метода всегда есть выражение из четырех слагаемых, получающееся при подстановке формы общего решения неоднородного уравнения в исходное дифференциальное уравнение (см. рис. 2, третья сторона обложки). В этом выражении реализован процесс дифференцирования, который неоднозначен по расположению сомножителей. Например, при дифференцировании степенной функции $(7x + 5)^3$ должно получиться $21(7x + 5)^2$, что соответствует эталонному решению. Вариант производной $3(7x + 5)^2 7$ обрабатывается без ошибок, но автоматически преобразуется к виду $21(7x + 5)^2$ после ввода 7.

Заключение

Большинство задач, решаемых студентами, например, в курсе высшей математики, относится к типовым задачам. В большей степени их решение связано с аналитическими преобразованиями. Эти преобразования нетрудно запрограммировать на языке высокого уровня. В данной статье для определенности в качестве языка программирования выбран язык Java, а в качестве программируемых преобразований выбраны решения линейных неоднородных дифференциальных уравнений методами Лагранжа и Бернулли-Фурье со всеми промежуточными действиями. Основой для иллюстрирующих программ, обучающих программ и программ-тренажеров является эталонное решение, преобразованное к графическому виду с полной информацией о каждом символе. Данное решение в различных приложениях используется по-разному. В иллюстрирующих и обучающих программах это решение посимвольно выводится на экран. В программах-тренажерах оно является информационным блоком для контроля правильности вводимых символов. Используется вариант контроля, в котором все ошибки являются причиной ввода лишних символов или не ввода недостающих символов. Для тренажера эталонное решение дополнено исключениями, обеспечивающими безошибочный ввод правильного решения, отличающегося от эталонного решения. Обсуждаемые программы моделируют последовательное написание символов на аудиторной доске или в тетради. Подобный подход, используемый при составлении компьютерных программ различного вида, может быть распространен на математические типовые задачи, полные решения которых нетрудно представить в виде алгоритмов для языков программирования высокого уровня.

Список литературы

1. **Клыков В. В., Ельцов А. А., Шатлов А. Г.** Интерактивные компьютерные тренажеры по интегральному исчислению и дифференциальным уравнениям // Известия Томского политехнического университета. 2006. Т. 309. № 2. С. 255—260.

2. **Мицель А. А., Клыков В. В.** Интерактивные компьютерные тренажеры по математическим дисциплинам // Открытое образование. 2005. № 2. С. 22—27.

3. **E-Math** — Тренажер по высшей математике. URL: http://e-math.ru/.

4. **Попов А. А.** Тренажер по нахождению первообразной функции для интеграла с дробно-иррациональным выражением // Вестник Марийского государственного университета. 2010. № 5. С. 297—300.

5. **Попов А. А.** Тренажер по аналитическому решению линейных дифференциальных уравнений методом Лагранжа // Материалы международной научно-практической конференции "Новые информационные технологии в образовании". Екатеринбург: Изд-во РГППУ. 2011. Ч. 1. С. 199—202.

6. **Демидович Б. П., Моденов В. П.** Дифференциальные уравнения. СПб.: Лань, 2006. 288 с.

7. **Ноутон Н., Шилдт Г.** Java 2: пер. с англ. СПб.: БХВ-Петербург, 2003. 1072 с.

ИНФОРМАЦИЯ

УДК 669.18.046.554:004.42

В. Д. Тутарова, канд. техн. наук, доц.,
Ю. В. Снегирев, аспирант, e-mail: y.v.snegirev@gmail.com
ФГБОУ ВПО "Магнитогорский государственный технический университет им. Г. И. Носова",

Повышение эффективности использования ресурсов вычислительной системы на примере решения задачи расплавления реагентов в сталеразливочном ковше

Рассматривается задача выбора средств для реализации расчета и визуализации расплавления реагентов в сталеразливочном ковше с точки зрения увеличения быстродействия и снижения потребления ресурсов вычислительной системы.

Ключевые слова: *математическое моделирование, внепечная обработка стали, теплофизические процессы, расплавление реагентов, выделение памяти, многопоточная обработка*

Актуальность

Современное металлургическое производство характеризуется высокой степенью автоматизации технологических процессов. Однако некоторые этапы производства высококачественной стали на настоящее время требуют обязательного участия в них человека. Следовательно, автоматизация труда операторов и повышение эффективности процессов управления такими технологически сложными агрегатами является актуальной задачей. Ее решение позволяет значительно снизить временные затраты, затраты на дорогостоящее сырье и энергоносители. Одним из таких агрегатов является установка печь-ковш, позволяющая оперативно осуществлять доведение характеристик стали (химический состав и температура) для непрерывной разливки до значений, диктуемых стандартами. В настоящий момент осуществление контроля с помощью отбора проб на химический анализ и измерение температуры сопряжены с дополнительными затратами времени, что значительно затрудняет управление реальным процессом. Разработка программного комплекса для прогнозирования физико-химических параметров расплава в режиме реального времени позволила бы существенно повысить скорость принятия решения о выполнении того или иного технологического этапа.

Постановка задачи

Для моделирования физико-химических процессов, протекающих при внепечной обработке стали на установке печь-ковш, был разработан программный комплекс "Моделирование процессов при внепечной обработке" [1].

При внедрении программного комплекса в действующее производство были предъявлены жесткие требования к быстродействию и потреблению ресурсов вычислительной системы. Поэтому одними из основных критериев реализации такого комплекса являлись минимизация времени моделирования и обеспечение работы комплекса в режиме реального времени.

Предлагаемое решение задачи

Разработка программного комплекса, удовлетворяющего требованиям, заложенным в модель, включает в себя разработку нескольких модулей как расчетных, так и представления данных результатов моделирования. Компоненты программного комплекса представлены UML-диаграммой (рис. 1).

Следует отметить, что данные из подсистемы "Расчет моделируемых параметров" можно также использовать для представления сводных результатов по работе каждого из модулей.

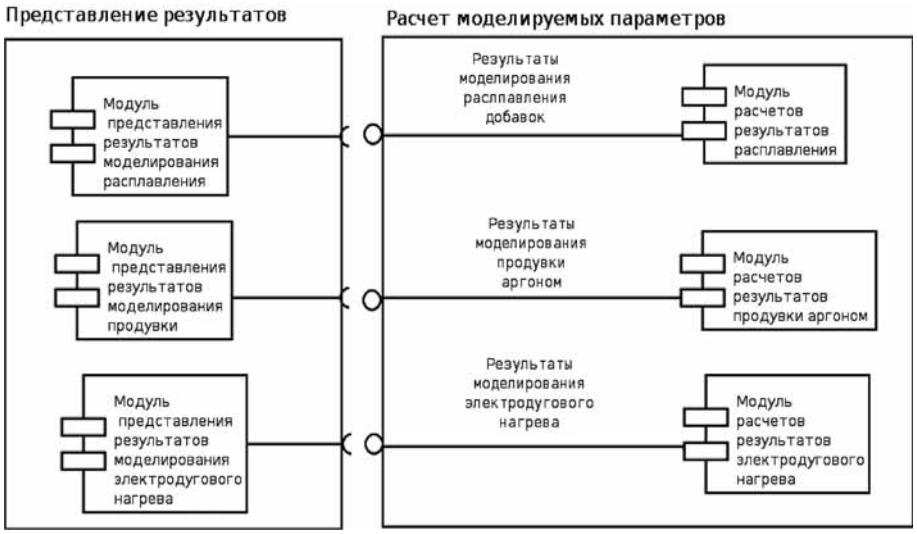


Рис. 1. Компоненты программного комплекса

В качестве средств реализации были проанализированы следующие программные продукты и методологии [2]:

- пакеты компьютерной алгебры MATLAB или SCILAB;
- язык программирования Python;
- библиотека на языке C или C++, которую можно было бы использовать в сочетании с уже зарегистрированной программой.

Критерием для их выбора послужили следующие характеристики:

- быстрота разработки и простота отладки;
- платформенная независимость;
- легкость интеграции с существующими АСУП и АСУТП.

Было решено остановиться на последнем варианте — реализовать модель на языке C. Предпочтение было отдано именно ему в силу того, что программы на C *проще поддаются переносу между платформами* (при грамотном проектировании интерфейсов модулей и использовании переносимых или стандартных библиотек). Кроме того, библиотеку, написанную на языке C, можно использовать в программах, написанных на *большом числе языков*. По сравнению с предыдущими вариантами, библиотека на языке C *обладает большим быстродействием,* а алгоритмы решения *лучше поддаются отладке вследствие более строгой типизации языка*. К тому же, из этой библиотеки при необходимости можно задействовать такие средства параллелизации вычислений, как OpenMP, MPI, CUDA и OpenCL.

Таблица 1

Работоспособность библиотеки

Архитектура	Операционная система		
	Linux	Windows	AIX
Intel x86	+	+	–
Intel x86_64	+	0	–
PowerPC 64	+	–	+

Примечание: + — работает; 0 — не проверялась; – — невозможно проверить.

Работоспособность библиотеки была протестирована на представленных в табл. 1 программных и аппаратных платформах, что свидетельствует о ее хорошей переносимости.

Для представления результатов были выбраны следующие средства:

- фреймворк Qt и язык C++;
- библиотека компонентов QWT;
- библиотека компонентов MathGL.

Фреймворк Qt был выбран в силу возможностей быстрого создания дружественных пользовательских интерфейсов; C++ — как его основной язык.

Температурное поле куска твердого реагента и температуру среды вокруг него целесообразно визуализировать спектрограммой.

Моделирование расплавления одного вида реагентов не приводит к затратам ресурсов вычислительной системы. Однако программный продукт должен позволять моделировать присадку *нескольких видов* реагентов *одновременно* — соответственно рассчитывать и визуализировать расплавление каждого из них. Это требование связано с проведением большого объема вычислений и может отрицательно сказаться как на отзывчивости[1] интерфейса пользователя, так и на производительности подсистемы в целом, что не согласуется с обозначенными ранее требованиями.

В указанных библиотеках (QWT и MathGL) содержится обширный набор компонентов для научных и технических приложений, одними из которых являются средства для построения спектрограмм. В случае быстрого обновления визуализируемых данных спектрограммы QWT значительно уступают по затратам процессорного времени на отрисовку спектрограммам MathGL.

Основными потребителями процессорного времени при построении спектрограмм являются: алгоритмы отрисовки, используемые в сторонних библиотеках; выделение памяти (аллокации); вычисления результатов моделирования.

Сократить время вычисления результатов моделирования можно за счет оптимизации алгоритмов.

Существует несколько групп способов сокращения времени на выделение памяти [3, 4]. Эти способы позволяют уменьшить фрагментацию "кучи" (*heap*) про-

[1] Отзывчивость интерфейса — время реакции интерфейса программного продукта на действия пользователя.

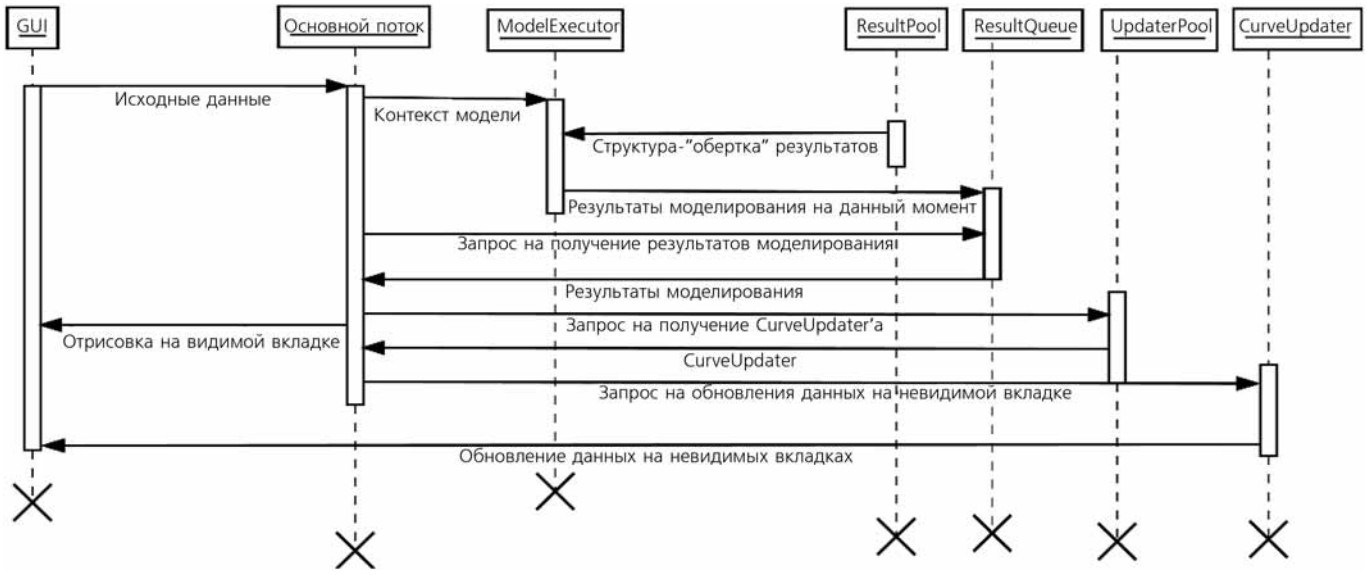


Рис. 2. Взаимодействие компонентов программного комплекса

цесса, что приводит к более рациональному расходу памяти, а также выделять память в пространстве пользователя — не происходит продолжительного системного вызова: ядро ОС не ищет среди списка доступных кусков фрагмент памяти необходимого размера.

Использование фреймворка Qt налагает ограничение на отрисовку элементов графического интерфейса пользователя: непосредственная отрисовка компонентов пользовательского интерфейса происходит на основном потоке приложения. Следовательно, в случае выполнения интенсивных вычислений на данном потоке это отрицательно сказывается на отзывчивости пользовательского интерфейса и производительности приложения.

По причинам, изложенным выше, целесообразно выделить интенсивные вычисления в отдельные вычислительные потоки и, по мере необходимости, передавать результаты в основной. Однако при этом возникают дополнительные трудности — целесообразно отрисовывать только спектрограмму, видимую в данный момент, и в то же время накапливать данные по изменению диаметров фаз и температур реагентов постоянно и для всех.

Таблица 2

Дополнительные потоки и их назначение

Поток	Назначение
ModelExecutor	Расчет результатов моделирования расплавления конкретной порции реагентов
CurveUpdater	Потоки обновления графиков постоянно изменяющихся характеристик

Таблица 3

Основные пуллы объектов и их назначение

Пулл	Назначение
ResultPool	Пулл "оберток" результатов вычислений
UpdaterPool	Пулл потоков CurveUpdater

В табл. 2 представлены дополнительные к основному потоки и их назначение.

Для уменьшения числа аллокаций используются пуллы объектов (табл. 3). Вместо того чтобы запрашивать память под результат моделирования у операционной системы, поток ModelExecutor, из которого необходимо передать данные в основной, запрашивает структуру, в которую "оборачивается" результат моделирования, из пулла и помещает ее в очередь.

Существует также пулл потоков, обновляющих графики размеров и температуры фаз реагентов. Использование данного пулла потоков сокращает расходы на аллокацию за счет выделения памяти как под структуры данных, так и под стеки потоков.

Взаимодействие компонентов программного комплекса представлено на рис. 2.

После ввода исходных данных в графическом интерфейсе пользователя (GUI) — вида реагента, его массы и начальной температуры; температуры стали в сталеразливочном ковше — запускается число потоков ModelExecutor, равное числу реагентов.

После завершения заданного числа итераций моделирования потоки ModelExecutor помещают результаты моделирования в очередь (ResultQueue), после чего каждый из потоков возобновляет работу.

Два раза в секунду на основном потоке происходит просмотр очереди, так как результаты моделирования "подписаны". При просмотре очереди результаты, не относящиеся к видимой спектрограмме, отправляются на обработку потоками CurveUpdater для соответствующей вкладки, а результаты для видимой вкладки сразу визуализируются.

* * *

Для сокращения времени визуализации результатов моделирования с помощью программного комплекса "Моделирование процессов при внепечной обработке" был осуществлен переход к библиотеке MathGL. При этом потребление CPU при отрисовке спектрограммы на видимой вкладке снизилось на 17 %.

Использование пуллов сократило число управляемых аллокаций в среднем на 60 единиц в секунду для каждого потока ModelExecutor.

Таким образом, выбранные средства позволяют значительно повысить быстродействие программного комплекса и снизить потребление ресурсов вычислительной системы.

В данной работе рассмотрена конкретная задача действующего производства как частный случай создания ресурсоемкого приложения для расчета и визуализации большого количества разнородных данных в режиме реального времени. Следует заметить, что предлагаемый подход применим в приложениях, в которых частота обновления визуализируемых данных высока, а отрисовка происходит от двух до десяти раз в секунду.

Список литературы

1. **Тутарова В. Д., Мацко И. И., Снегирев Ю. В., Калитаев А. Н.** Моделирование процессов при внепечной обработке: Свидетельство о государственной регистрации программы для ЭВМ № 2010615314. — 2010613706. Зарег. 18.08.2010. ОБПБДТ. 2010. № 4. С. 252.

2. **Снегирев Ю. В., Тутарова В. Д.** Моделирование расплавления реагентов в сталеразливочном ковше при внепечной обработке // Приволжский научный вестник. 2011. № 2. С. 16—22.

3. **Bonwick J.** The Slab Allocator: An Object-Caching Kernel Memory Allocator (1994). Sun Microsystems — 1994. [Электронный ресурс]. Систем. требования: просмотрщик PostScript. URL: http://www.usenix.org/publications/library/proceedings/bos94/full_papers/bonwick.ps.

4. **Sen A., Kardam R. K.** Building your own memory manager for C/C++ projects. IBM Developer Works. URL: http://www.ibm.com/developerworks/aix/tutorials/au-memorymanager/index.html.

ИНФОРМАЦИЯ

Уважаемые коллеги!
Приглашаем Вас к авторскому участию в журнале "Программная инженерия"!

МАТЕРИАЛЫ, ПРЕДСТАВЛЯЕМЫЕ В РЕДАКЦИЮ

- Статья, оформленная в соответствии с требованиями.
- Иллюстрации и перечень подрисуночных подписей.
- ФИО авторов, название статьи, аннотация и ключевые слова на английском языке.
- Сведения об авторах (ФИО, ученая степень, место работы, занимаемая должность, контактные телефоны, e-mail).

ПОРЯДОК ОФОРМЛЕНИЯ ИНФОРМАЦИОННОЙ ЧАСТИ

- Индекс ***УДК*** размещается в левом верхнем углу первой страницы.
- ***Сведения об авторах*** на русском языке размещаются перед названием статьи и включают инициалы и фамилию авторов с указанием их ученой степени, звания, должности и названия организации и места ее расположения (если это не следует из ее названия). Указывается также e-mail и/или почтовый адрес хотя бы одного автора или организации.
- За сведениями об авторах следует ***название статьи***.
- После названия статьи отдельным абзацем дается ***краткая аннотация***, отражающая содержание статьи (что в ней рассмотрено, приведено, обосновано, предложено и т.д.).
- Затем следуют ***ключевые слова***.

ТЕКСТ СТАТЬИ

Статью рекомендуется разбить на разделы с названиями, отражающими их содержание. Рекомендуемый объем статьи 15 страниц текста, набранного на стандартных листах формата А4 с полями не менее 2,5 см шрифтом размером 14 pt с полуторным межстрочным интервалом, с использованием компьютерного текстового редактора Word. В указанный объем статьи включаются приложения, список литературы, таблицы и рисунки. Страницы статьи должны быть ***пронумерованы***. В необходимых случаях по решению главного редактора или руководства издательства объем статьи может быть увеличен. В формулах русские и греческие буквы следует набирать прямо; латинские буквы, обозначающие скаляры, *курсивом*; величины, обозначающие векторы и матрицы, должны быть выделены полужирным шрифтом и набраны прямо (допускается также набор всех величин, обозначенных латинскими буквами, в т.ч. матриц и векторов, светлым курсивом). Стандартные математические обозначения (например, log, sin и т.д.) должны быть набраны прямо. Номера формул располагаются справа в круглых скобках. Нумерация формул — сквозная, причем нумеруются те формулы, на которые имеются ссылки.

Рисунки должны быть выполнены качественно (графическая обработка рисунков в редакции не предполагается). В журнале все рисунки воспроизводятся в черно-белом варианте, за исключением цветных рисунков, размещаемых по усмотрению редакции на обложке. Статья может быть отправлена по e-mail: prin@novtex.ru

Дополнительные пояснения авторы могут получить в редакции журнала лично, по телефонам:
(499) 269-53-97, 269-55-10, либо по e-mail.

CONTENTS

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 07.09.2012 г. Подписано в печать 18.10.2012 г. Формат 60×88 1/8. Заказ PI812
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
105120, г. Москва, ул. Нижняя Сыромятническая, д. 5/7, стр. 2, офис 2.