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 004.457

А. Н. Годунов, канд. физ.-мат. наук., зав. отд.,
В. А. Солдатов, канд. техн. наук., стар. науч. сотр., НИИСИ РАН, г. Москва,
e-mail: nkag@niisi.ras.ru

Особенности отладки встроенных систем

Рассматриваются методы мониторинга и отладки встроенных систем на различных этапах их жизненного цикла. Описываются инструментальные средства, входящие в состав отечественной операционной системы реального времени ОСРВ Багет 2.0, предназначенные для сбора информации о ходе выполнения системы и сохранения ее на имеющихся носителях.

Ключевые слова: *ОСРВ Багет-2.0, встроенные системы, отладка, системный журнал, протоколирование, исключительная ситуация*

Под встроенной системой понимается программно-аппаратный комплекс, функционирующий без вмешательства человека и обычно являющийся частью более крупной распределенной системы управления [1]. Встроенные системы, как правило, работают в режиме жесткого реального времени, а целевые модули, на которых они выполняются, имеют ограниченные ресурсы.

Так как встроенные системы обычно используют для управления жизненно важными объектами, то к надежности их функционирования и восстановлению работоспособности после сбоя предъявляются высокие требования. Для повышения надежности в программном обеспечении должны содержаться средства, контролирующие ход выполнения программы и реагирующие на различные внештатные ситуации.

В силу ограниченности ресурсов целевого модуля программирование встроенных систем зачастую выполняется на инструментальной ЭВМ под управлением операционной системы общего назначения, т. е. используется технология кросс-разработки [2].

На первом этапе разработки встроенная система выполняется на стенде, который представляет собой математическую или полунатурную модель [3] реального объекта, и лишь на втором этапе разработки можно переходить к полнонатурным испытаниям системы на реальном объекте. Для отладки программ во встроенных системах на обоих этапах используют следующие основные методы [4]:

- интерактивная отладка, допускающая вмешательство пользователя в работу отлаживаемой системы;
- мониторинг, при котором в хронологической последовательности с различной степенью детализации собирают сведения о происходящих в системе событиях (ошибки, предупреждения, сообщения).

В свою очередь, интерактивная отладка бывает:

— удаленной (кросс-отладка), в терминах исходного языка программирования, когда отладчик выполняется на инструментальной ЭВМ, а его агент — на целевом модуле;

— локальной, в терминах ассемблера, когда весь диалог с пользователем выполняется на целевом модуле с помощью команд локального отладчика под управлением командного интерпретатора.

В мониторинге можно выделить:

— протоколирование событий средствами, встроенными в операционные системы реального времени (ОСРВ) в формате, предназначенном для последующего просмотра и анализа результатов с помощью интерактивной программы "Трассировщик", выполняемой на инструментальной ЭВМ [5];

— протоколирование событий с помощью функций вывода текстовых сообщений из прикладных программ пользователя [6];

— средства самоконтроля целевых и системных программ [7];

— отчет об исключительной ситуации (*crash report*) [8].

Удаленная кросс-отладка, как правило, применяется только при выполнении встроенной системы на стенде, так как на реальном объекте для этого вида отладки нет достаточных ресурсов. Использование на реальном объекте диалоговых средств локальной отладки напрямую зависит от наличия на этом объекте консоли (постоянной или временной). Использование программы "Трассировщик" и полного дампа памяти (*core dump*) в качестве отчета об исключительной

ситуации также возможно только при наличии достаточных ресурсов, т. е. на первом этапе разработки и отладки встроенной системы.

К сожалению, не все ошибки выявляются в процессе отладки встроенной системы на стенде. Часть ошибок впервые обнаруживается уже при испытаниях на реальных объектах. Нередки случаи, когда ошибки проявляются в непредсказуемый момент, после многих часов или суток безотказной работы системы и только на реальном объекте. Для выявления и локализации таких ошибок основным средством являются протокол работы системы и отчет об исключительной ситуации.

Необходимо отметить, что даже если все испытания успешно пройдены, это не всегда означает, что ошибки не могут проявиться в ходе дальнейшей эксплуатации системы. По этой причине приходится систематически собирать сведения о функционировании системы, в том числе и об отказах, и периодически анализировать полученные протоколы.

При выполнении встроенной системы на реальном объекте существуют значительные ограничения как на инструментальные средства, необходимые для удаленной отладки, так и на ресурсы, доступные для запоминания информации. Во встроенных системах зачастую отсутствуют связь с инструментальной ЭВМ, возможность иметь оператора — человека, который может анализировать поведение системы, а также не всегда возможно подключение консоли, жестких дисков. Таким образом, получается, что на различных этапах жизни программного обеспечения встроенных систем не только используются различные инструментальные средства отладки, но и вывод сообщений должен быть организован различным образом. На начальных этапах, когда программы выполняются на стенде, в присутствии разработчика, вывод сообщений удобно направлять на консоль. В дальнейшем, при опытной и производственной эксплуатации вдали от разработчика, необходимым и важным условием становится поддержка механизмов сбора и хранения выводимых сообщений. В качестве запоминающего устройства для реализации этих целей можно использовать диск, флеш-память или другое запоминающее устройство. Периодически вся собранная информация считывается с носителя и передается для анализа разработчикам. Переход от одного способа организации вывода к другому не должен требовать никаких изменений в системе, кроме указания конфигуратору требуемого адресата вывода.

Следует также учитывать, что вывод диагностических сообщений необходимо обеспечить и в случае ошибок при обработке прерываний от внешних устройств или при зависании операционной системы. Для вывода таких сообщений можно задействовать самые простые механизмы из имеющихся в операционной системе. Таковым является вывод сообщений во флеш-память или, что еще проще, в область оперативной памяти, которая сохраняет свое значение при перезагрузке модуля. В случае использования оперативной или флеш-памяти и наличия диска при каждой перезагрузке сообщения предыдущего сеанса переписываются на диск. В случае отсутствия постоянного диска сообщения могут быть переписаны на переносной диск с помощью соответствующей утилиты.

Рассмотрим средства ОСРВ Багет 2.0 [9], предназначенные для решения некоторых из перечисленных выше задач.

Протоколирование событий с помощью системного журнала

Для вывода текстовых сообщений в ОСРВ Багет 2.0 могут использоваться различные функции с интерфейсом, аналогичным printf(). Следует однако учитывать, что не все эти функции можно применять в обработчике прерываний, а также в экстремальных ситуациях, например, при запрещенном планировании.

Наиболее универсальной функцией вывода сообщений является функция syslog(), предназначенная для вывода текстовых сообщений в системный журнал, которую можно использовать из любых контекстов выполнения ОСРВ Багет 2.0. Интерфейс этой функции соответствует стандарту POSIX, но при ее реализации были добавлены некоторые дополнительные возможности, которые позволяют организовать вывод не только в файл или на консоль, но и в заданную область флеш-памяти или в область оперативной памяти, которая не стирается при перезагрузке.

Сообщение, формируемое функцией syslog(), состоит из заголовка и содержательной части — тела. Заголовок сообщения содержит время, уровень и источник сообщения. Тело сообщения соответствует формату, который определяет пользователь при вызове функции. Первый параметр функции указывает на уровень важности сообщения, второй определяет формат тела выводимого сообщения. Далее следует переменное число аргументов, тип и количество которых определяется вторым параметром. При обращении к функции syslog() можно в первом параметре добавить указание выводить сообщение без заголовка.

Рассмотрим алгоритм реализации функции syslog(). Вывод текстовых сообщений с помощью этой функции обычно выполняется системой в два этапа. На первом из них отформатированное сообщение помещается в буфер оперативной памяти. Далее находящиеся в буферах сообщения выводятся специальным высокоприоритетным потоком (syslog-демоном). Адресат вывода сообщения задается при конфигурировании целевого модуля.

Все буфера системного журнала имеют одинаковую длину. Буфер может находиться в очереди свободных буферов или в очереди буферов с сообщениями

на вывод. Обычно функция syslog() извлекает элемент из очереди свободных буферов, переносит в него выводимое сообщение и помещает этот элемент в выходную очередь сообщений. Если сообщение не помещается целиком в буфер, то оно укорачивается до длины буфера. Если очередь свободных буферов пуста, сообщение теряется, однако счетчик потерянных сообщений при этом увеличивается на единицу. При установке элемента в пустую выходную очередь сообщений функция syslog() активизирует поток syslog-демон, который атомарно извлекает все находящиеся в выходной очереди сообщения, счетчик потерянных сообщений и обнуляет значение этого счетчика. Далее поток syslog-демон выводит сообщения в той последовательности, в которой они были помещены в очередь. После вывода очередного сообщения соответствующий ему элемент помещается в очередь свободных буферов. После вывода последнего сообщения выводится значение счетчика потерянных сообщений, если оно отлично от нуля. Если выходная очередь сообщений пуста, поток syslog-демон переходит в состояние ожидания до тех пор, пока в выходной очереди не появятся сообщения.

Приоритет, с которым выполняется поток syslog-демон, размер одного буфера и число буферов системного журнала задаются при конфигурировании целевого модуля. Кроме того, разработчику встроенной системы необходимо определить, куда именно будут выводиться сообщения системного журнала. Этот параметр может принимать следующие значения:

- console — сообщения выводятся на консоль модуля;
- file — сообщения записываются в файл;
- flash_memory — сообщения сохраняются в заданной области флеш-памяти;
- persistent_memory — сообщения переносятся в заданную область оперативной памяти модуля, которая сохраняет свое значение при перезагрузке модуля;
- user_functions — вывод сообщений осуществляется функцией, заданной пользователем.

Если был определен вывод сообщений не на консоль модуля, то можно заказать дублирование сообщений системного журнала на консоль. В случае, когда определен вывод сообщений в файл, дополнительно необходимо задать имя этого файла. Если файл с указанным именем не существует, он будет создан, а если файл уже существует, то сообщения системного журнала будут выводиться в конец этого файла.

Если сообщения системного журнала выводятся во флеш-память, то необходимо определить адрес начала области флеш-памяти и ее размер. Указанная область предварительно должна быть очищена. После того как указанная область флеш-памяти будет полностью заполнена, сообщения начнут выводиться на консоль или, если у системы нет консоли, будут потеряны.

Если сообщения системного журнала выводятся в сохраняемую при перезагрузке оперативную память модуля, то дополнительно надо задать имя области сохраняемой памяти, ее размер и определить действия системного журнала в случае, когда вся область заданной памяти будет заполнена. Возможны следующие два варианта:

- не поместившиеся сообщения перенаправляются на консоль модуля;
- сообщения системного журнала записываются в заданную область памяти как в кольцевой буфер.

В случаях, когда определен вывод сообщений системного журнала во флеш-память или в сохраняемую при перезагрузке область оперативной памяти и есть жесткий диск, то для исключения потери ранее сохраненных сообщений можно определить файл, в который будут перемещены все сообщения системного журнала при перезагрузке модуля. Если нет жесткого диска, то выгрузку накопленных сообщений можно проводить по команде на переносное запоминающее устройство. Возможны перечисленные далее три различных способа задания файла на жестком диске.

- Определяется имя файла. Сообщения каждого сеанса начинают выводиться в конец файла с заданным именем. Перед началом вывода сообщений очередного сеанса будет записана строка специального вида, содержащая порядковый номер сеанса вывода сообщений. Первая строка файла содержит строку специального вида с номером последнего сеанса записи сообщений в этот файл.
- Определяется имя файла и его максимально допустимый размер. В этом случае вывод сообщений в файл проводится аналогично выводу в циклический буфер, а именно — по достижении максимально допустимого размера сообщения начинают записываться в начало файла. Начало файла содержит строку специального вида с указателем на текущее место для записи сообщений в файл и номером последнего сеанса записи сообщений в файл. Перед началом вывода порции сообщений каждого очередного сеанса также записывается строка специального вида, в которой содержится номер сеанса.
- Определяется корень имени файла (<корень>). Имя файла для вывода сообщений получается с помощью конкатенации этого корня с суффиксом вида ".<*nnn*>", где <*nnn*> — это трехзначный порядковый номер файла с указанным корнем. Очередное значение суффикса, которое он может принимать — от 001 до 999, автоматически формируется системой. Если все файлы заполнены, то сообщения очередного сеанса начинают выводиться в конец файла с именем <корень>.<999>.

Пользователь может задать вывод сообщений с помощью собственных функций вывода. В этом случае надо определить имя пользовательской функции вывода сообщений. Дополнительно могут быть заданы имена функций открытия и закрытия для вывода сообщений.

При выполнении встроенной системы могут случаться разного рода ошибки, например исключительная ситуация (*exeption*). Если исключительная ситуация произошла при запрещенном планировании, или в контексте обработчика прерываний, или в контексте потока syslog-демон, то будем констатировать, что произошла тяжелая ошибка. При обнаружении тяжелой ошибки устанавливается специальный флаг "тяжелая ошибка", который влияет на вывод сообщений в системный журнал. Если установлен флаг "тяжелая ошибка", то вывод сообщения осуществляется уже не в контексте потока syslog-демон, а непосредственно в том контексте, в котором произошла ошибка, так как в этом случае невозможно переключение контекста на поток syslog-демон.

Из контекста обработчика прерываний могут вызываться не все функции вывода. Однако вывод системных сообщений во флеш-память, в память, значение которой не стирается при перезагрузке, а также на консоль в операционной системе ОСРВ Багет 2.0 возможен в любом контексте. Здесь следует отметить, что в случае тяжелой ошибки вывод на консоль возможен благодаря тому, что операционная система организует его по алгоритму, отличному от алгоритма вывода в обычных случаях, а именно происходит опрос устройства и не используется механизм прерываний. Если же был определен вывод сообщений в файл, то в случае тяжелой ошибки вывод будет перенаправлен на консоль модуля, а если консоли нет, сообщения будут потеряны. Такое поведение системного журнала связано с отсутствием возможности выполнять функцию записи в файл при запрещенном планировании и из контекста обработчика прерываний. Если был определен вывод сообщений пользовательской функцией, то ответственность за возможность выполнения этой функции в случае тяжелой ошибки лежит на пользователе.

Описанная в работе схема реализации вывода в системный журнал позволяет накапливать в автоматическом режиме сообщения системного журнала на внешних носителях за достаточно большое число сеансов работы системы в целях их дальнейшего анализа.

Обработка ошибочных ситуаций

Если формирование отчетов об ошибочных ситуациях осуществлять средствами вывода сообщений в системный журнал, то все отчеты могут накапливаться на носителе, указанном при конфигурации системы, например, во флеш-памяти. Ошибочные ситуации, которые могут возникнуть при выполнении встроенных систем, можно разделить на следующие группы:

- ошибки, обнаруженные средствами самоконтроля;
- исключительные ситуации;
- зависание системы.

При зависании системы единственным методом восстановления ее работоспособности является перезагрузка. Во встроенных системах, когда рядом нет человека, который может нажать кнопку reset, для этого используется аппаратно реализованная схема контроля за зависанием системы. Эта схема называется сторожевым таймером (*watchdog timer*) и представляет собой таймер, заряженный на некоторый интервал времени, который периодически должен сбрасываться системой. Если до истечения заданного интервала времени система не сбросила сторожевой таймер, то происходит ее перезагрузка. В зависимости от настройки сторожевой таймер может программно или аппаратно перезагружать целевой модуль.

В случае ошибки, обнаруженной средствами самоконтроля, обычной реакцией системы является формирование краткого отчета об обнаруженном сбое в работе системы и восстановление ее работоспособности в зависимости от ситуации, вплоть до перезагрузки.

В случае возникновения исключительных ситуаций стандартными средствами системы формируется отчет об ошибочной ситуации и, в зависимости от конфигурации, происходит перезагрузка системы или приостановление выполнения ошибочного потока.

Рассмотрим разделы отчета об ошибочной ситуации, которые могут формироваться стандартными средствами при возникновении на целевом модуле исключительной ситуации. Перечень разделов, входящих в состав отчета, указывается при конфигурировании целевого модуля.

1. Первый раздел отчета выводится всегда и содержит основные сведения, диагностирующие ошибочную ситуацию, а именно: наименование ошибки и номер вектора исключительной ситуации (*Error*); адрес команды, выполнение которой вызвало исключительную ситуацию (*Address*); идентификатор потока, при выполнении которого произошла ошибка (*Thread*); реакция системы на возникшую исключительную ситуацию (*Action*).

Все остальные разделы отчета выводятся только в том случае, если вывод раздела был определен при конфигурировании модуля. Перечислим назначение каждого из них.

2. Содержимое регистров общего назначения.

3. Если ошибочная ситуация произошла при выполнении команд сопроцессора плавающей арифметики и вывод этого раздел был указан при конфигурировании модуля, то выводятся значения регистров сопроцессора для обработки плавающих чисел.

4. Вывод окрестности команды, при выполнении которой произошла исключительная ситуация, на языке ассемблер (дизассемблирование исполняемого кода).

5. Вывод стека вызовов функции, при выполнении которой произошла исключительная ситуация.

6. Список потоков, которые выполнялись на целевом моде, и подробная информация о потоке, при выполнении которого произошла ошибка.

7. При формировании отчета может быть проведена проверка на корректность всех системных объектов операционной системы. В отчет помещается протокол, содержащий перечень ошибочных объектов с указанием класса объекта и типа обнаруженной ошибки, или сообщение об отсутствии ошибочных объектов.

8. Вывод содержимого оперативной памяти заданного размера, на которую указывает регистр стека, так называемый стековый кадр (*stack frame*).

9. Сообщения, которые сформировала пользовательская функция.

Все разделы отчета об ошибочной ситуации выводятся средствами системного журнала, что обеспечивает их сохранность на носителе, заказанном при конфигурировании целевого модуля для их последующего анализа.

Описанные выше средства протоколирования с сохранением результатов на внешних носителях получили широкое распространение среди пользователей ОСРВ Багет 2.0. Приведем один из примеров, случившихся в реальности. При испытаниях системы управления на реальном объекте, вдали от разработчиков, все данные о каждом запуске собирались во флеш-памяти. Последующий анализ собранной информации показал, что иногда среди диагностических сообщений о ходе испытаний встречается отчет об ошибочной ситуации:

```
* Error     :  TLB exception   (load or instr. fetch) (2)
* Address   :  0x80142830
* Thread    :  0x81bebe58
* Action    :  REBOOT
* Nestlevel : 2

epc=0xffffffff80142830 cause=0x00050008 status=0x34010003
bva=0x000000000000001e
sp=0xffffffff804a1f30 fp=0xffffffff804a1f30 ra=0xffffffff80154138
s0=0xffffffff800306c4 s1=0x0000000000000000 s2=0x0000000000000000
s3=0x0000000000000000
s4=0x0000000000000000 s5=0x0000000000000000 s6=0x0000000000000000
s7=0x0000000000000000
a0=0xffffffff80450000 a1=0x000000000000001e a2=0x000000000000ffff
a3=0x0000000000000001
v0=0x000000000000001e v1=0x000000000000001e
t0=0x0000000034010000 t1=0x0000000000000002 t2=0x0000000000000000
t3=0x0000000000000000
t4=0x0000000000000000 t5=0x0000000000000000 t6=0x0000000000000000
t7=0xffffffff8027b2f0
STOP AREA (0x80142830):
_acRegRead+68:
80142820:   00621021   addu  $v0,$v1,$v0
80142824:   97c30018   lhu   $v1,24($s8)
80142828:   8c420018   lw    $v0,24($v0)
8014282c:   00621021   addu  $v0,$v1,$v0
80142830:   94420000   lhu   $v0,0($v0)
80142834:   a7c20000   sh    $v0,0($s8)
80142838:   97c20000   lhu   $v0,0($s8)
8014283c:   304200ff   andi  $v0,$v0,0xff
80142840:   00021a00   sll   $v1,$v0,0x8
80142844:   97c20000   lhu   $v0,0($s8)
BACK TRACING:
801ba920 kernThreadStub+90    : main (0x0)
800309f0 main+6c              : RegimLoad (&disp_35, &kod_znak_sy_35)
80034ce8 RegimLoad+3c         : WorkUVV ()
80034c8c WorkUVV+10           : read_key ()
80034000 read_key+25c         : mkioMfiPutData (2, 65536, 0x8046c502)
80140374 mkioMfiPutData+54    : aceRTDataBlkWrite (1, 1179647, 0x8046c502)
801618a4 aceRTDataBlkWrite+4a8: _acMemBlockWrite (49848, 439016, 0x8046c502)
801424dc _acMemBlockWrite+ec  : -->()
801a6de0 vecHandler+1b8       : vecHandlerStub ()
801a6f6c vecHandlerStub+40    : 800146b4 ()
80014700 pciConfigInLong+158  : mmkHardIntProc (0)
80154654 mmkHardIntProc+320   : _acRegRead (0, 0, 0x8001)
80142830: 94420000 lhu $v0,0($v0)
```

Как видно из представленного листинга, ошибочная ситуация случается в обработчике прерываний, на что указывает уровень прерываний (Nestlevel=2) и стек вызовов функций (—>). Реакция, предусмотренная в системе на эту ситуацию, — перезагрузка, которая с учетом восстановления полной работоспособности занимает в данной системе около семи секунд. Анализ исходного кода программы, относящегося к ошибочной ситуации, показал, что такое возможно только в случае неправильного значения указателя на структуру, предназначенную для хранения параметров прибора. Учитывая, что подобная ситуация не воспроизводилась на испытательном стенде, а на реальном объекте случалась только на одной из испытуемых установок, был сделан вывод о том, что имеет место сбой в работе оборудования. Действительно, на этой установке был обнаружен скрытый дефект в блоке питания и, после исправления обнаруженного недостатка, случаи с ошибочной ситуацией больше не наблюдались.

В заключение еще раз отметим, что для встроенных систем, которые выполняются вдали от разработчиков и в отсутствие оператора, важное значение приобретают средства сбора информации о выполнении системы и регистрации сбоя в их работе. Инструментальные средства, входящие в состав ОСРВ Багет 2.0, позволяют собирать информацию о выполнении и сбоях в работе систем и запоминать ее на имеющихся носителях.

Список литературы

1. **Баррет С. Ф., Пак Д. Дж.** Встраиваемые системы. Проектирование приложений на микроконтроллерах семейства 68HC12/HCS12 с применением языка С. М.: ДМК Пресс, 2007.

2. **Саммерфилд М.** Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++. М.: Символ-Плюс, 2011.

3. **Овчинников М. Ю., Ткачев С. С.** Компьютерное и полунатурное моделирование динамики управляемых систем. Препринт ИПМ им. М. В. Келдыша РАН, 2008.

4. **Костюхин К. А.** Обзор методов и средств отладки распределенных систем // Информационная безопасность. Микропроцессоры. Отладка сложных систем / под ред. В. Б. Бетелина. М.: НИИСИ РАН, 2004. С. 76—119.

5. **Годунов А. Н., Жихарский Л. В., Назаров П. Е., Чемерев Ф. Н.** Средства протоколирования в ос2000 // Инструментальные средства программирования / под ред. В. Б. Бетелина. М.: НИИСИ РАН, 2006. С. 84—111.

6. **Карпов А.** Построение систем автоматического протоколирования Си/Си++ кода. URL: http://www.viva64.com/ru/a/0023/

7. **Костюхин К. А.** Средства самоконтроля программ и их применение при отладке систем со сложной архитектурой // Информационная безопасность. Микропроцессоры. Отладка сложных систем / под ред. В. Б. Бетелина. М.: НИИСИ РАН, 2004. С. 151—160.

8. **CrashRpt** Documentation. URL: http://crashrpt.sourceforge.net/docs/html/ index.html.

9. **Безруков В. Л., Годунов А. Н., Назаров П. Е., Солдатов В. А., Хоменков И. И.** Введение в ос2000 // Вопросы кибернетики / под ред. В. Б. Бетелина. М.: НИИСИ РАН, 1999. С. 76—106.

ИНФОРМАЦИЯ

30—31 мая 2013 г., в г. Казань состоится

7-й Весенний коллоквиум молодых ученых по программной инженерии SYRCOSE-2013,

организованный Институтом системного программирования РАН, Санкт-Петербургским государственным университетом и Казанским национальным исследовательским университетом им. А. Н. Туполева при поддержке Intel, Российского фонда фундаментальных исследований и Нижегородского фонда содействия образованию. Коллоквиум приглашает к участию в работе студентов, аспирантов и молодых кандидатов наук (до 35 лет).

Более подробная информация доступна на сайте мероприятия: http://syrcose.ispras.ru/

УДК 004.054

А. С. Лысунец, нач. отдела, Главное управление Центрального банка Российской Федерации по Санкт-Петербургу, e-mail: las366@spb.cbr.ru
Б. А. Позин, д-р техн. наук, техн. директор, ЗАО "ЕС-лизинг", г. Москва

Планирование контроля целостности сложной автоматизированной банковской системы методами функционального и нагрузочного тестирования

Изложены проблемы контроля целостности сложных автоматизированных банковских систем при сопровождении. Дано понятие уровня целостности автоматизированной системы, описаны использующие оценку уровня целостности методика планирования тестирования и методика оценки адекватности его результатов, а также методика оценки эффективности контроля целостности автоматизированной банковской системы.

Ключевые слова: *сопровождение автоматизированной системы, целостность автоматизированной системы, уровень целостности, планирование контроля целостности, регрессионное нагрузочное тестирование*

К организации контроля целостности при сопровождении автоматизированных банковских систем

Рассматриваемые в настоящей статье автоматизированные банковские системы (АБС) принадлежат к классу транзакционных. Такие системы архитектурно представляют собой сложный аппаратно-программный комплекс, функционирующий в режиме 24×7. С точки зрения процессов жизненного цикла программных средств, в соответствии с ГОСТ 12207—99 [1] АБС уже в течение ряда лет находятся в процессе эксплуатации и сопровождения. Целью процесса сопровождения является изменение программного обеспечения АБС при сохранении его целостности [2].

Для систем рассматриваемого класса характерно достаточно частое изменение нормативных документов и совершенствование бизнес-процессов, автоматизируемых средствами АБС. Вносимые в АБС корректировки в подавляющем большинстве случаев необходимы для дополнения (изменения) функций АБС и исправления ошибок, выявленных в процессе эксплуатации. На рис. 1 приведена статистика по формированию заданий на доработку АБС, в которых приводятся новые или уточняются существующие требования к функциям и сроки их реализации. Как правило, каждое задание определяет реализацию одного требования. В случае, когда для автоматизации

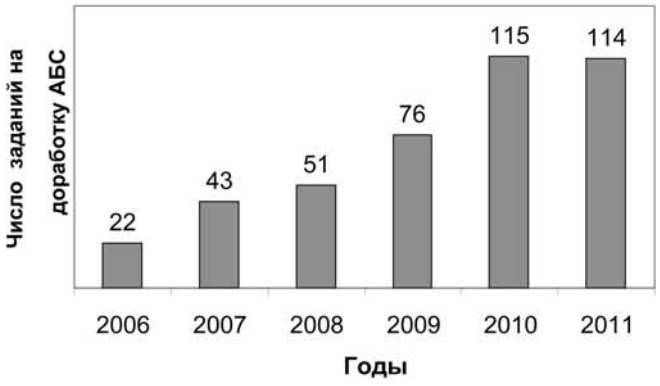


Рис. 1. Статистика по формированию заданий на доработку АБС

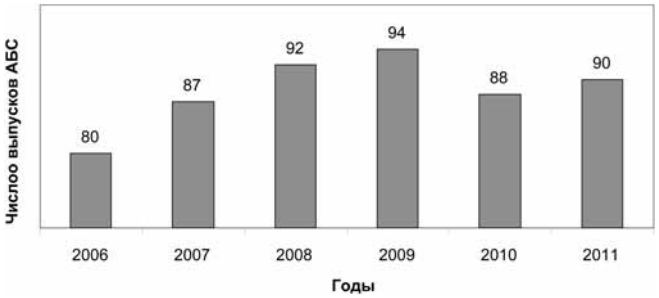


Рис. 2. Статистика по выпускам ППО АБС

бизнес-процесса необходима реализация нескольких требований, выпускается набор связанных заданий. Это особенно характерно для существенных изменений в АБС. Основанием для модификации прикладного программного обеспечения (далее — ППО) АБС в том числе являются и заявки пользователей. По статистике общее число таких заявок, подаваемых за год, составляет около 2000, из них примерно половина реализуется в АБС.

В процессе реализации заданий на доработку АБС и заявок пользователей в ППО АБС вносятся соответствующие изменения. Все изменения, выполненные к определенному моменту времени, включаются в перечень, представляющий собой выпуск (релиз) ППО АБС. Каждому выпуску присваивается порядковый номер, уточняются сроки его ввода в действие. На рис. 2 приведена статистика по числу выпусков (релизов) ППО АБС (далее — выпуск АБС).

Число изменений, включаемых в каждый выпуск АБС, для таких систем в течение года варьируется от 200 до 1500. Часть изменений могут носить глобальный характер, когда модификации подвергаются основные, системообразующие функции АБС. По статистике в течение года может возникать несколько (до 2—4) изменений такого типа.

Ввод в действие выпусков АБС после внесения изменений осуществляют не ежедневно, а с некоторой периодичностью, так что на контроль целостности системы после внесения изменений отводится не более 5—7 дней. Целью такого контроля является обнаружение ошибок в модифицированной АБС, поэтому основным методом его реализации является регрессионное тестирование. Тестирование проводится для выявления как нарушений при выполнении функциональных требований (регрессионное функциональное тестирование), так и деградации ожидаемых эксплуатационных характеристик АБС (регрессионное нагрузочное тестирование) [5].

Регрессионное тестирование всегда проводится по заранее подготовленным тестам. Для рассматриваемого класса систем число тестов, необходимых для проверки работоспособности АБС в целом, составляет несколько тысяч. Вследствие существенной сложности АБС как объекта контроля и ограничения ресурсов календарного времени возникают вопросы планирования и создания методики организации работ по контролю целостности системы методами тестирования. Контроль целостности АБС осуществляется, как правило, одним и тем же коллективом специалистов, независимо от применяемых методов тестирования (функциональное или нагрузочное). По этой причине наиболее приемлемой представляется некая унифицированная методика планирования обоих видов тестирования. Такая методика должна быть эффективной как при незначительных изменениях, так и при значимых модификациях АБС.

Понятие уровня целостности АБС

Определим объект контроля, деятельность по оценке которого предполагается планировать. Согласно ГОСТ 15026—2002 [3] под уровнем целостности понимается "указание диапазона значений свойства объекта, необходимых для удержания системного риска в допустимых границах. Для объектов, отказ которых может привести к угрозе для системы, таким свойством является ограничение частоты подобного отказа".

Представленное выше определение полностью подходит для случая АБС. Для того чтобы сделать уровень целостности измеримым, рассмотрим следующую модель. Пусть система S представляет собой структуру, состоящую из нескольких взаимодействующих элементов (подсистем), для каждого из которых можно задать или измерить вероятность p_i выполнения подлежащей контролю функции или значение соответствующей эксплуатационной характеристики. Тогда можно оценить P_s — интегральную вероятность выполнения заданного комплекса функций, например, методами теории надежности. Значение P_s отражает свойство системы выполнять возложенные на нее задачи в процессе эксплуатации или, согласно приведенному выше определению, текущий уровень ее целостности.

Может существовать **заданный уровень целостности**, т. е. такой уровень целостности, который требуется заказчику или эксплуатирующей объект организации. Этот факт означает, что в любой момент времени система выполняет возложенные на нее задачи с определенной вероятностью и риски нарушения бизнес-процессов устраивают заказчика или эксплуатирующую организацию. Достижение заданного уровня целостности при эксплуатации системы необходимо не всегда. Причиной этого может быть, например, разная степень востребованности функций системы в течение некоторого периода времени. С точки зрения целевой деятельности организации, использующей систему в своих интересах, может быть установлен **допустимый уровень целостности**. Таким является уровень, при котором использование системы еще возможно, т. е. угрозы производственному процессу еще приемлемы, однако они максимальны.

Внесение модификаций в систему в процессе сопровождения изменяет уровень ее целостности в зависимости от количества и качества этих модификаций. С этих позиций **контроль целостности системы** представляет собой оценку уровня целостности модифицированной программной системы на соответствие заданному или допустимому уровню ее целостности.

Методика планирования контроля целостности АБС и оценки адекватности его результатов

Контроль целостности необходимо проводить для каждого выпуска АБС. Существенный дефицит ресурсов (в основном календарного времени) делает необходимым организацию **процесса планирования контроля целостности АБС** при ее сопровождении. Этот процесс заключается в разработке программы тестовых испытаний, включающей перечень планов проверки требований, предъявляемых к АБС, и последовательность выполнения этих планов. По этой же причине программа тестовых испытаний должна соответствовать следующим требованиям (стратегии):

- программа должна позволять при имеющихся ресурсах сопроводителя (тестировщика) максимизировать достижимый уровень целостности модифицированной системы методами функционального и нагрузочного тестирования, постоянно сохраняя его не ниже допустимого, по возможности обеспечивая при этом лучшее приближение к заданному уровню целостности;
- одна и та же программа тестирования с минимальными корректировками должна позволять эффективно контролировать целостность практически для любого выпуска АБС, независимо от объемов и характера изменений.

При контроле целостности в процессе сопровождения проверке подвергаются так называемые бизнес-сценарии — последовательности действий по использованию АБС, описывающие, как правило, реализацию одного функционального требования с точки зрения пользователя или заказчика.

Бизнес-сценарии являются объектами проверки как при функциональном, так и при нагрузочном тестировании. Разница состоит в целях тестирования. При функциональном тестировании необходимо проверить сам факт выполнения бизнес-сценария. Целями проверки при нагрузочном тестировании могут быть: проверка того, что бизнес-сценарий выполняется в установленные регламентом сроки; оценка деградации эксплуатационных характеристик, например, среднего времени выполнения бизнес-сценария.

Разным бизнес-сценариям соответствуют различные уровни риска нарушения при эксплуатации АБС. Эти уровни риска могут характеризоваться следующими показателями:

- критичность бизнес-сценария — степень опасности последствий, которые могут наступить в случае отказа в работе АБС при выполнении бизнес-сценария;
- вероятность использования бизнес-сценария — частота выполнения бизнес-сценария при эксплуатации АБС.

Уровни значений этих показателей для критичности и вероятности использования бизнес-сценариев устанавливаются совместно с заказчиком или эксплуатирующей организацией. Редко удается установить количественные значения этих показателей, чаще они выражаются качественно. Например, критичность бизнес-сценария может оцениваться как фатальная, высокая, допустимая, незначительная, а вероятность использования — как обычная, возможная, маловероятная.

Риск неблагоприятного исхода бизнес-сценария (далее — просто риск) определяется с использованием матрицы рисков, связывающей вероятность использования бизнес-процесса с опасностью его неблагоприятных последствий. Матрица риска устанавливает уровни риска, такие как критичный, высокий, средний, низкий, несущественный. В таблице приведены уровни риска бизнес-процессов для АБС.

Числу установленных уровней риска соответствует число уровней целостности АБС. Ответственный за контроль целостности совместно с заказчиком или эксплуатирующей организацией должен определить, каким уровням риска соответствует допустимый и заданный уровни целостности.

В случае с АБС модифицированная система соответствует допустимому уровню целостности, если выполняются бизнес-сценарии с критичным и высоким уровнями риска. Для того чтобы модифицированная АБС соответствовала заданному уровню целостности, необходимо проверить выполнение всех бизнес-сце-

Уровни риска бизнес-процессов для АБС

Вероятность использования бизнес-сценария	Опасность последствий при нарушении бизнес-сценария			
	Фатальная	Высокая	Допустимая	Незначительная
Обычная	Критичный	Критичный	Высокий	Средний
Возможная	Критичный	Высокий	Средний	Низкий
Маловероятная	Высокий	Высокий	Средний	Несущественный

нариев с уровнем риска от критичного до низкого включительно.

Автоматизируемым в АБС бизнес-процессам свойственна цикличность применения. Одни из них используются в течение всего рабочего дня или только в определенные часы, другие — по завершении декады, месяца, квартала, года. Критичность бизнес-сценариев остается неизменной на всем периоде эксплуатации АБС, тогда как вероятность использования может варьироваться в зависимости от периода календарного времени, в течение которого АБС эксплуатируется. Бизнес-сценарии, имеющие в одни и те же периоды эксплуатации АБС одинаковые уровни риска, группируются в типовые наборы. Использование наборов, определяющих стандартную программу тестовых испытаний для каждого уровня целостности в зависимости от периода эксплуатации АБС, позволяет оценивать число необходимых для тестирования ресурсов. Практика показывает, что внесение изменений в типовые наборы бизнес-сценариев необходимо только при глобальных изменениях в АБС. А такие изменения происходят не чаще 2—3 раз в год.

В целях качественного внесения изменений в АБС при ее сопровождении организован процесс управления выпусками, в ходе которого на основе поступивших на доработку заданий и заявок пользователей определяется расписание выхода выпусков АБС. Процесс управления выпусками позволяет определять период времени, в течение которого будет эксплуатироваться каждый выпуск АБС. В результате заранее известна программа тестовых испытаний, позволяющая имеющимися силами (персоналом) и за выделенное время оценить уровень целостности практически любой версии АБС. Ожидается, что уровень целостности при этом должен соответствовать максимально возможному уровню, но быть не ниже допустимого.

Отличия в целях функционального и нагрузочного тестирования при контроле целостности являются причиной того, что подходы к проектированию планов проверок выполнения бизнес-сценариев (планов функционального и нагрузочного тестирования) различаются.

Модель бизнес-сценария можно представить в виде ориентированного графа, вершинами которого являются функциональные элементы АБС, необходимые для выполнения этого бизнес-сценария. Ребра графа определяют возможные переходы между функциональными элементами. Для каждой вершины графа указан точный результат, ожидаемый при работе функционального элемента. Число вершин и ребер заранее известно.

План функционального тестирования представляет собой наиболее эффективный маршрут обхода всех вершин графа. Этот маршрут, как правило, предусматривает проверку одного бизнес-сценария. По этой причине в случае изменения бизнес-сценария в план тестирования могут быть оперативно внесены необходимые корректировки. Результатом функционального тестирования является дискретная величина, однозначно определяющая в любой момент времени факт выполнения бизнес-сценария. Проверки по планам функционального тестирования могут выполняться (в общем случае) независимо друг от друга, так как каждый бизнес-сценарий описывает выполнение одного функционального требования.

При нагрузочном тестировании оценивается влияние внесенных в систему изменений на способность выполнения бизнес-сценария за отведенное, регламентное время. При эксплуатации системы на одной и той же технической платформе происходит одновременное выполнение большого числа бизнес-сценариев. На эксплуатационные характеристики бизнес-сценария оказывают влияние как другие бизнес-сценарии, так и объем и состав данных, обрабатываемых системой при выполнении ею бизнес-сценариев. В разные периоды времени работы системы реализуемые ею бизнес-сценарии могут иметь различные значения эксплуатационных характеристик. Получаемые при нагрузочном тестировании результаты представляют собой вероятностные величины, имеющие математическое ожидание и дисперсию. Они необходимы для оценки деградации эксплуатационных характеристик бизнес-сценариев, выполняемых модифицированной АБС. По результатам выполнения плана нагрузочного тестирования делается вывод о способности системы в целом, а не отдельных ее программных или аппаратных компонентов, обеспечить выполнение всех бизнес-сценариев в течение эксплуатации выпуска АБС. Для того чтобы такой вывод можно было сделать, при разработке плана нагрузочного тестирования решаются задачи, связанные: с выбором расчетных показателей, характеризующих эксплуатационные характеристики бизнес-сценариев; с планированием нагрузки, учитывающей выполнение взаимодействующих и взаимовлияющих бизнес-сценариев; с определением состава испытательного стенда, моделирующего работу АБС; с описанием процедуры измерений. Подробно их решение изложено в работе [4].

План нагрузочного тестирования для выполнения контроля целостности АБС должен соответствовать следующим требованиям:

- на подготовку, проведение и оценку результатов нагрузочного тестирования должно хватать времени, отведенного для контроля целостности выпуска АБС. При этом необходимо учитывать, что нагрузочное тестирование должно проводиться после функционального тестирования бизнес-сценариев, для которых оценивается деградация эксплуатационных характеристик [5];
- по результатам нагрузочного тестирования должна существовать возможность оценить деградацию эксплуатационных характеристик бизнес-сцена-

риев, включенных в план контроля целостности выпуска АБС.

Выполнение этих требований обеспечивается за счет моделирования работы АБС в наиболее экстремальных, но потенциально возможных в ходе эксплуатации условиях. Для оптимизации затрат на подготовку и проведение экспериментов используется система автоматизированного нагрузочного тестирования.

Для выявления признаков деградации эксплуатационных характеристик бизнес-сценариев появляется необходимость в хранении результатов нагрузочного тестирования для статистической обработки данных по проведенным ранее экспериментам [4]. Как следствие, нагрузочное тестирование каждого выпуска АБС должно проводиться в одних и тех же условиях, по одному и тому же плану. Влияние вносимых в АБС изменений на план нагрузочного тестирования должно быть сведено к минимуму. В случае существенных изменений подконтрольной системы в процессе управления ее выпусками необходимо предусмотреть ресурсы на корректировку и отладку плана нагрузочного тестирования.

Конечным требованием к результатам нагрузочного тестирования АБС является их адекватность результатам, которые фиксируются (наблюдаются) в ходе ее эксплуатации. Очевидным способом контроля выполнения указанного требования является сравнение результатов, полученных при нагрузочном тестировании каждого выпуска АБС, с результатами, полученными для того же выпуска системы при ее эксплуатации. Неоспоримое преимущество этого способа заключается в возможности внесения корректировок в план нагрузочного тестирования в случае разницы результатов. Однако в силу того что формирование выпусков АБС идет "непрерывно" во времени, применение способа сравнения результатов имеет важное ограничение. Сложность реализации этого способа состоит в том, что тестирование следующего выпуска АБС проходит в то время, когда результатов эксплуатации предыдущего выпуска системы еще нет. В таких условиях задача обеспечения адекватности результатов нагрузочного тестирования АБС решается на основе сбора и анализа статистики.

По статистическим данным, накопленным по результатам нагрузочного тестирования каждого выпуска АБС, формируются средние значения и дисперсии расчетных показателей эксплуатационных характеристик реализации бизнес-сценариев. Полученные значения сопоставляются со сведениями о состоянии успешности эксплуатации системы для уточнения максимально допустимых значений показателей. Для каждого эксперимента проводится анализ попадания результатов тестирования в диапазон допустимых значений показателей эксплуатационных характеристик. С увеличением числа проведенных экспериментов точность статистических данных повышается при условии, что план нагрузочного тестирования не изменяется.

Методика оценки эффективности планирования контроля целостности АБС

В процессе эксплуатации очередного выпуска АБС могут быть выявлены ошибки, а именно — невозможность выполнить какой-либо бизнес-сценарий (комбинацию сценариев) или получение эксплуатационных характеристик бизнес-сценария ниже допустимых значений. Причины того, что нарушение работоспособности АБС не было обнаружено до передачи ее в эксплуатацию, могут быть связаны с ошибками, допущенными как при составлении (корректировке) программы тестовых испытаний, так и при формировании планов проверок выполнения бизнес-сценариев. Возникает необходимость вносить изменения в планирование контроля целостности. Важно, чтобы эти изменения не снизили эффективность планирования. При выполнении контроля целостности по измененной программе тестовых испытаний должны быть обнаружены не только ошибки, выявленные в ходе эксплуатации текущего выпуска АБС и послужившие причиной корректировки. Модифицированная программа тестовых испытаний должна продолжать выявлять ошибки, обнаруженные в предыдущих выпусках АБС.

Самый надежный способ оценить эффективность контроля целостности АБС по измененной программе тестовых испытаний заключается в том, чтобы последовательно выполнить программу для всех предыдущих выпусков системы. Практика показывает, что необходимые ресурсы отсутствуют даже для повторного контроля целостности по скорректированной программе последнего проверенного выпуска АБС [6].

Очевидно, что эффективность планирования контроля целостности напрямую зависит от эффективности программы тестовых испытаний, которая, в свою очередь, зависит от наличия ошибок, выявленных в ходе эксплуатации системы. Если в процессе эксплуатации выпуска АБС не будет выявлено ошибок в выполнении бизнес-сценариев требуемого уровня целостности, то программа тестовых испытаний для данного выпуска АБС является эффективной. Однако существуют и другие исходы.

В случае выявления ошибок в процессе эксплуатации выпуска АБС нельзя однозначно говорить о неэффективности программы тестовых испытаний. Эта неопределенность возникает, так как при контроле целостности, выполненном по "неэффективной" программе, могут быть обнаружены и устранены ошибки, оказывающие влияние на целостность АБС. При оценке эффективности программы тестовых испытаний необходимо проводить совокупный анализ результатов контроля целостности и результатов эксплуатации, полученных для одного и того же выпуска АБС.

Критерием эффективности программы тестовых испытаний может являться соотношение числа ошибок, выявленных при контроле целостности для выпуска АБС, к суммарному числу ошибок, выявленных

для того же выпуска АБС при эксплуатации и при контроле целостности [6]. Сравнение выполняется для ошибок при выполнении бизнес-сценариев, относящихся к требуемому, как правило, к допустимому уровню целостности.

Величину R_i будем называть успешностью тестирования для i-го выпуска АБС:

$$R_i = \frac{X_i}{X_i + Y_i},$$

где X_i — число ошибок, выявленных при контроле целостности АБС, для i-го выпуска АБС; Y_i — число ошибок, выявленных при эксплуатации АБС, для i-го выпуска АБС.

Существуют следующие предельные случаи:

- $R_i = 1$ — означает, что все ошибки были выявлены при контроле целостности выпуска АБС. В этом случае можно говорить о том, что программа тестовых испытаний способствует обнаружению всех ошибок и является предельно эффективной. К этому результату необходимо стремиться при планировании контроля целостности для каждого выпуска АБС.
- $R_i = 0$ — означает, что при контроле целостности выпуска АБС ошибок не было обнаружено, а в процессе эксплуатации они проявились. В этом случае можно говорить о наличии существенных просчетов, допущенных при планировании контроля целостности. Такая программа тестовых испытаний является абсолютно неэффективной.
- $X_i + Y_i = 0$ — означает, что ошибки не были выявлены ни при контроле целостности выпуска АБС, ни при эксплуатации. В этом случае можно говорить о том, что при внесении изменений не было оказано влияние на целостность АБС, работа по корректировке системы выполнена качественно, эффективность программы тестовых испытаний оценке не подлежит.

В остальных случаях, когда ошибки выявляются и при контроле целостности, и при эксплуатации одного и того же выпуска АБС, нельзя однозначно оценить эффективность программы тестовых испытаний. Необходим анализ поведения R_i на интервале от 0 до 1:

$$0 < \frac{X_i}{X_i + Y_i} < 1.$$

Очевидно, если число ошибок, выявленных при контроле целостности АБС, меньше, чем число ошибок, выявленных при эксплуатации того же выпуска АБС ($X_i < Y_i$ или $R_i < 0{,}5$), то программу тестовых испытаний нельзя назвать эффективной.

Остается рассмотреть варианты, когда число ошибок, выявленных при контроле целостности АБС, больше, чем число ошибок, выявленных при эксплуатации того же выпуска АБС ($0{,}5 \leqslant R_i < 1$).

В идеальном случае значение R_i должно каждый раз быть больше R_{i-1} и стремиться к 1. Такая ситуация будет означать, что каждое внесение корректировок в программу тестовых испытаний, выполненное по результатам эксплуатации АБС, положительно влияет на ее эффективность. Однако на практике значение R_i не растет линейно.

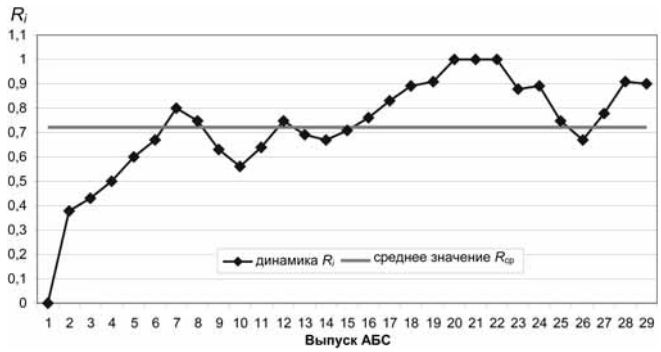


Рис. 3. Пример графика успешности тестирования выпусков ППО АБС R_i

Рассмотрим пример графика успешности тестирования выпусков ППО АБС R_i, показанный на рис. 3.

Для выпуска 1 АБС успешность тестирования R_1 оказалась равной 0, т. е. программа тестовых испытаний, по которой проводился контроль целостности, не обеспечила обнаружение ошибок, допущенных в АБС при внесении изменений. По результатам анализа при планировании контроля целостности следующего выпуска АБС в программу тестовых испытаний были внесены необходимые корректировки, позволившие повысить успешность тестирования. Однако программа тестовых испытаний по-прежнему не позволила выявить ошибок больше, чем было выявлено при эксплуатации выпуска 2 АБС. Такое положение дел в данном примере сохранялось до выпуска 4 АБС, когда программа тестовых испытаний позволила выявить половину всех ошибок.

Дальнейшая корректировка программы тестовых испытаний привела к росту успешности тестирования для выпусков 5, 6 и 7. После выпуска 7 начался спад эффективности программы тестовых испытаний для выпусков 8, 9 и 10. Данный спад произошел по причине отсутствия качественных планов, необходимых для проверки модифицированных в этих выпусках АБС бизнес-сценариев. Начиная с выпуска 11 эффективность программы тестовых испытаний снова начала расти, так как к этому моменту были подготовлены и отлажены планы функционального и нагрузочного тестирования бизнес-сценариев, набрана статистика для оценки их эксплуатационных характеристик.

Похожая динамика эффективности программы тестовых испытаний была для последующих выпусков 13—19, а также для выпусков 23—28.

Для выпусков 20, 21 и 22 эффективность программы тестовых испытаний достигла своего максимального значения. Методика планирования контроля целостности позволила обнаружить все ошибки, спо-

собные привести к сбоям в работе АБС при эксплуатации.

В связи с высокой интенсивностью внесения изменений в АБС удержать эффективность программы тестовых испытаний на протяжении большого числа выпусков затруднительно. Однако если для какого-либо выпуска АБС не была достигнута максимальная эффективность, это не значит, что планирование контроля целостности выполняется неэффективно. В практике контроля целостности АБС интервал значений R_i, на котором считается, что проведенный контроль целостности был эффективным, определен от среднего значения успешности тестирования, рассчитанного для всех выпусков АБС ($R_{ср}$), до 1. В данном примере $R_{ср} = 0{,}72$, т. е. если 72 % всех ошибок для версии АБС будет обнаружено при контроле целостности, то планирование выполнено эффективно. Значение нижнего порога эффективности можно задать и более жестко, например, равным 0,85—0,9 или 80—90 %.

Выводы

Понятие целостности автоматизированной системы становится ключевым, когда после создания система передается в эксплуатацию и начинается процесс ее сопровождения. Для обеспечения целостности системы необходима соответствующая организация всех этапов ее сопровождения. Должны быть разработаны не только качественные, но и количественные метрики и критерии всего процесса, включая отдельные методы и методики (например, автоматизированного тестирования). В настоящей работе сделан один из первых шагов в этом направлении для ППО АБС. При сопровождении автоматизированной системы качество ее функционирования во многом определяется способностью соответствующих специалистов эффективно планировать контроль целостности системы после внесения в нее изменений.

В течение ряда лет на основе опыта практических работ по сопровождению и тестированию АБС разработана модель целостности и методика планирования контроля ее целостности. В основе такой методики лежит оценка уровня риска бизнес-сценариев, которые должна выполнять система в одни и те же периоды эксплуатации, и группировка бизнес-сценариев в типовые наборы с одинаковым уровнем риска. Типовые наборы определяют стандартную программу тестовых испытаний в зависимости от периода эксплуатации АБС. В работе приведена стратегия составления программы тестовых испытаний. Обязательным условием применения методики является планирование выпусков системы при сопровождении.

Методика позволяет планировать контроль целостности как при функциональном, так и при нагрузочном тестировании. В работе особое внимание уделено подготовке нагрузочного тестирования АБС. На основе опыта работ по контролю целостности определены требования к составлению сценария регрессионного нагрузочного тестирования и оценке адекватности результатов проверок. В соответствии с данными требованиями для каждого выпуска системы обеспечивается достоверная оценка деградации эксплуатационных характеристик АБС.

При применении изложенной в статье методики планирования контроля целостности крайне важной является возможность оценки ее эффективности. Разработанный подход позволяет вести постоянный контроль за состоянием планирования контроля целостности АБС в условиях, когда в силу высокой трудоемкости тестирования и жестких ограничений на календарное время проведения тестирования отсутствует возможность оценить эффективность корректировок программы тестовых испытаний путем повторного контроля целостности одного и того же выпуска системы.

Разработанная методика планирования контроля целостности АБС и оценки эффективности планирования в совокупности с процессом планирования выпусков АБС обеспечивают удовлетворяющие заказчика результаты эксплуатации АБС не только при незначительных изменениях, но и при значимых модификациях системы. На практике удается достигнуть средней эффективности планирования контроля целостности не ниже 96—98 % в условиях появления новых выпусков АБС через каждые 5—7 дней и выполнения тестирования коллективом специалистов не более 10 человек.

Список литературы

1. **ГОСТ 12207—99.** Информационная технология. Процессы жизненного цикла программных средств. М.: ИПК Издательство стандартов, 2000.

2. **ГОСТ 14764—2002.** Информационная технология. Сопровождение программных средств. М.: ИПК Издательство стандартов, 2002.

3. **ГОСТ 15026—2002.** Информационная технология. Уровни целостности систем и программных средств. М.: ИПК Издательство стандартов, 2002.

4. **Позин Б. А., Галахов И. В.** Модели в нагрузочном тестировании // Программирование. 2011. № 1. С. 20—35.

5. **Позин Б. А.** Ввод в действие информационных систем и сопровождение их программного обеспечения // Информационные технологии. 2010. № 4. Приложение. 36 с.

6. **Лысунец А. С.** Планирование контроля целостности банковских систем при сопровождении в условиях ограниченности ресурсов // Материалы II Научно-практической конференции "Актуальные проблемы системной и программной инженерии", Сборник научных трудов. М.: Издательство МЭСИ, 2011. С. 196—201.

УДК 004.42

Е. А. Кольчугина, канд. техн. наук, Государственное образовательное учреждение высшего профессионального образования Пензенский государственный университет,
e-mail: kea_sci@list.ru

Программные системы с самоорганизацией континуального типа

Рассматриваются виды изменяющихся программ, в том числе самоорганизующиеся программные системы. Выделены виды и условия самоорганизации. Рассмотрены модели теории искусственной химии, в которых исследуются процессы самоорганизации программ. Анализируется концепция неравновесного программирования, предложенная в работах автора. Определены перспективы развития технологий проектирования и использования самоорганизующихся программных систем.

Ключевые слова: *разработка программного обеспечения, парадигма программирования, самоорганизация, искусственная химия, неравновесное программирование*

Введение

Одна из основных проблем, возникающих при создании полезных на практике и эффективных программ, состоит в необходимости постоянного поддержания таких программ в состоянии, которое гарантирует их соответствие требованиям заказчика при внесении объективно необходимых изменений в уже ранее созданный код.

Причины этого следующие:

- на стадии разработки технического задания и проектирования не всегда удается сразу выделить и описать все необходимые функции программного обеспечения (ПО), предусмотреть все возможные условия его функционирования (например, состояние вычислительной среды), возможные ситуации и способы реакции на них;
- требования к ПО, в том числе к реализуемым функциям, к качеству их реализации, к условиям функционирования, к реакциям на возникающие события, могут со временем изменяться в силу как объективных, так и субъективных (например, изменение пожеланий пользователя) причин. Необходимость постоянного пересмотра и совершенствования программных продуктов на всех этапах их жизненного цикла, в том числе внедренных в рамках уже выполненных проектов, привела к появлению спиральной модели ПО [1], а также других современных инженерных технологий разработки и сопровождения ПО.

Отличительными чертами современных методов и технологий разработки ПО являются [1—8]:

- высокий уровень абстрагирования;
- возможность описания решения задачи в терминах, близких предметной области;
- объектная и событийная ориентированность;
- модульность и применение компонентного подхода, что является предпосылкой использования комбинаторности;
- унифицируемость и стандартизация программных решений, тенденция к повторному использованию кода, в том числе широкое применение шаблонов для генерации кодов, метапрограммирование, порождающее программирование и фабрики ПО;
- интеграция с базами данных и знаний;
- кроссплатформенность, применение средств middleware и виртуальных машин для разработки и выполнения переносимых программ;
- распределенность вычислений и данных в компьютерных сетях;
- параллелизм обработки информации;
- комплексность как превращение программ в коллективы агентов или многоуровневые стратифицированные системы (приложения из приложений).

Как правило, совершенствование продуктов, уже созданных в рамках программных проектов, предполагает пересмотр и переписывание программного кода, т. е. возврат к стадиям пересмотра требований, реализации и тестирования. Более прогрессивным представляется подход, при котором программа могла бы перенастраивать или модифицировать, полностью или частично, реализуемые ею алгоритмы уже на стадии эксплуатации благодаря вмешательству человека или без такового.

Программы, обладающие такой способностью, будем называть **изменяющимися**. Итак, изменяющиеся программы — это программы, которые каким-либо образом могут изменять свой код или свое поведение на этапе эксплуатации в целях совершенствования, которыми могут быть: повышение практической полезности; расширение функциональности; повышение эффективности вычислений; адаптация к вычислительному окружению, а также индивидуальным особенностям и потребностям пользователя.

Виды изменяющихся программ

В зависимости от способности к изменению и степени участия человека в процессе изменения программы на стадии эксплуатации будем различать следующие виды изменяющихся программ:

- адаптивное ПО;
- динамически расширяемое ПО;
- самомодифицирующееся и полиморфное ПО;
- эволюционирующее ПО;
- самоорганизующееся ПО.

Рассмотрим каждый вид подробнее.

Адаптивное программное обеспечение — ПО, способное к изменению порядка выполнения начально заданного алгоритма и/или порядка вычислений на основе настройки параметров (с учетом конкретного пользователя, решаемой задачи, вычислительного окружения), изменения данных или знаний.

В рамках этого вида ПО будем различать:

- настраиваемое ПО, которое адаптируется на основе изменения параметров конфигурации;
- интеллектуальное ПО, которое изменяет порядок выполнения начально заданного алгоритма и/или порядок вычислений на основе изменения состояния среды окружения баз данных и знаний о проблемной области, а также в результате накопления и формирования новых данных и знаний, дедуктивного вывода на знаниях. К разновидностям последнего можно отнести программное обеспечение, реализованное в виде коллективов агентов [6], системы автоматизированного динамического распараллеливания [9].

Динамически расширяемое программное обеспечение — ПО, способное к расширению функциональных возможностей путем добавления новых программных единиц в процессе эксплуатации. Такое расширение достигается путем обращения основной расширяемой программы к дополнительным программным модулям (plug-in, add-ons), например, в соответствии с моделью "клиент-сервер".

Самомодифицирующееся программное обеспечение — ПО, способное изменять либо свой исходный код, который впоследствии подвергается интерпретации, либо исполняемый код в процессе выполнения [10]. Данный подход применяется в компьютерных играх, web-приложениях.

Эволюционирующее программное обеспечение — ПО, способное к развитию и оптимизации непосредственно самого программного кода в результате процессов, аналогичных дарвиновской эволюции. Основой для реализации программ этого вида являются следующие технологии:

- эволюционное программирование [11];
- генетическое программирование [12—14];
- рекомбинантное программирование [15].

Эволюция программ этого вида может происходить как в процессе проектирования и разработки, так и в процессе эксплуатации.

Эволюционное программирование было предложено в работе Фогеля и Уолша [11]. Оно сходно с генетическими алгоритмами и иногда рассматривается как их непосредственный предшественник. Сущность данного метода состоит в том, что случайным мутациям и отбору подвергается кодированное представление автомата, включающее описание набора правил перехода и выдачи выходных символов. В настоящее время этот метод применяется для формирования стратегий поведения интеллектуальных агентов (аниматов).

Генетическое программирование — метод разработки программ, основанный на направленном переборе. Метод относится к эволюционным вычислениям, сходен с генетическими алгоритмами. Программные решения представляют собой хромосомы особей, из которых выбираются наиболее подходящие по заданному критерию приспособленности. Хромосомы могут описываться как древовидными структурами на языках обработки списков (λ-исчисление) [12, 13], так и линейными последовательностями ассемблерных команд или групп команд (μ-исчисление) [14].

Рекомбинантное программирование — парадигма программирования, рассматривающая образование программы из отдельных последовательных фрагментов кода путем рекомбинации и склейки [15], подобно тому, как цепочки ДНК могут образовываться из меньших цепочек путем "сшивки" "липких" концов [16]. Базовым для данной парадигмы является понятие последовательности, состоящей из оснований, или основных функциональных элементов. С основаниями могут быть связаны атрибуты, называемые "пигментами". Группы "пигментов" образуют "цвета", которые вместе с логическими условиями ветвления используются в процессе рекомбинации для соединения отдельных последовательностей. Парадигма рекомбинантного программирования была реализована в языке Grapple, ее общими недостатками являются технологическая сложность и неочевидность сфер применения.

В дальнейшем будем рассматривать самоорганизующееся ПО. Чтобы дать определение этому виду ПО, рассмотрим понятие самоорганизации как таковой.

Самоорганизация — это способность системы (системного объекта) к внутреннему самоупорядочиванию. В соответствии с определением, данным Г. Хакеном [17], самоорганизация означает способность без специфического воздействия извне приобретать структуру: временную, пространственную, функциональную.

Самоорганизация, таким образом, предполагает невозрастание энтропии или даже ее уменьшение. В соответствии со вторым началом термодинамики, энтропия в замкнутой системе возрастает и достигает максимума. Однако в открытых системах, обменивающихся веществом и энергией с окружающей средой, возможно невозрастание и даже уменьшение энтропии благодаря экспорту энтропии во внешнюю среду [18—23]:

$$dS = dS_i + dS_e,$$

где dS_i — производство энтропии внутри системы; dS_e — поток энтропии. Если $dS \leqslant 0$ и так как $dS_i > 0$, то $dS_e \leqslant 0$, $|dS_i| \leqslant |dS_e|$.

Следовательно, ***самоорганизующееся программное обеспечение***:

— является системным объектом, обладающим структурой;

— способно к упорядочиванию и переупорядочиванию своих элементов и связей, т. е. образованию и изменению структуры;

— открытое;

— способно к отводу энтропии во внешнюю среду.

Необходимыми свойствами самоорганизующихся систем являются также способность совершать работу против равновесия со средой (в первую очередь термодинамического равновесия) и наличие свободной энергии [23]. Эти свойства будут подробнее рассмотрены далее.

Будем различать два вида самоорганизующихся программных систем.

Полностью самоорганизующаяся программная система — это система, которая образуется и существует благодаря процессам самоорганизации, проходящим в специальным образом устроенной среде. В такой системе процессы самоорганизации присутствуют на всех последующих этапах жизненного цикла, так как система не изымается из породившей ее среды [24].

Программная система, полученная путем самоорганизации, отличается от полностью самоорганизующейся системы тем, что по достижении ею какого-либо состояния, удовлетворительного с точки зрения проектировщиков или конечных пользователей, она извлекается из среды, в которой была получена, и в дальнейшем процессов самоорганизации в ней не происходит, система фиксируется в том состоянии, "как она есть" [24].

Следовательно, главными вопросами при создании самоорганизующихся программных систем являются:

- принципы построения сред, в которых могут происходить процессы самоорганизации;
- определение минимальных неделимых далее программных единиц и правил их компоновки в структуры более высоких порядков;
- определение правил, регулирующих процессы самоорганизации и задающих критерии оптимальности решения.

Условия и виды самоорганизации

В настоящее время выделяют два основных принципа структурной организации вещества [23]:

- принцип организации, для которого характерно стремление к равновесию со средой, минимуму внутренней энергии и максимуму энтропии;
- принцип самоорганизации, для которого характерно стремление к неравновесию, обмен веществом-энергией со средой, уменьшение энтропии благодаря ее отводу во внешнюю среду и стремление к увеличению внутренней энергии, которая расходуется на совершение работы против равновесия.

Необходимыми условиями для проявления внутрисистемной самоорганизации и образования эволюционирующих структур являются [20—23]:

- избыточность и конкуренция структур;
- ограниченность времени существования структур;
- неравновесность и антиэнтропийная направленность процессов;
- мультистабильность, неустойчивость стационарных состояний и удаление от равновесия в случае малых возмущений;
- открытость и диссипативность системы;
- наличие свободной энергии;
- нелинейность математических моделей, описывающих взаимодействия элементов макросистем друг с другом (на уровне микросистем возможна линейность [22]).

Еще одним необходимым условием является слаженность поведения элементов в составе микро- или макросистем. В теории эволюционного катализа [23] различают два типа самоорганизации, связанных с видом и системными уровнями проявления слаженности взаимодействия при образовании структур: когерентный (синергетический) и континуальный (синкретический).

Когерентная (синергетическая) самоорганизация основана на кооперативности поведения однородных микросистем в составе макросистемы [23]. Данный тип самоорганизации приводит к синхронизации параметров микросистем и пространственно-геометрической упорядоченности, но прогрессивных изменений при данном типе самоорганизации не происходит [23], возможна только линейная эволюция [22, 23].

Континуальный тип самоорганизации рассматривается в теории эволюционного катализа и предпола-

гает взаимодействие разнородных элементов на уровне микросистем. Решающее значение для данного типа самоорганизации имеет образование в ходе базисной реакции промежуточного неравновесного функционально неделимого объекта, а именно кинетического континуума веществ и реакций [23]. Этот тип самоорганизации приводит к функциональной упорядоченности и прогрессивной эволюции. Более того, континуальная самоорганизация определяет возможность когерентной самоорганизации.

Следовательно, модель программной системы, в которой спонтанно происходит образование самоорганизующихся эволюционирующих структур, должна быть, во-первых, компонентной, во-вторых, подобной моделям эволюционного катализа, с образованием неравновесного континуума компонентов и взаимодействий между ними. Из компьютерных моделей наиболее близко таким требованиям отвечают модели теории искусственной жизни [25, 26], а именно такая их разновидность, как модели искусственной химии [27, 28].

Модели искусственной химии

Целью исследований в области теории искусственной жизни является воспроизведение принципов динамики, присущей живым системам, в виртуальной компьютерной среде [25, 26]. Одним из направлений, существующих в рамках этой теории, являются модели искусственной химии [27, 28].

В соответствии с данным в работе [27] определением, модель искусственной химии есть тройка вида (S, R, A), где S — множество возможных типов молекул; R — набор правил столкновения, представляющих взаимодействие между молекулами; A — алгоритм, описывающий камеру химического реактора, т. е. пространственную область, в которой происходят реакции, и то, как применяются правила к молекулам внутри химического реактора.

Первой моделью искусственной химии была модель алгоритмической химии AlChemy, предложенная У. Фонтана [28]. В этой модели роль молекул выполняли функции, которые можно рассматривать и как программы на λ-подобном языке, и как данные, в случае если функция является нуль-арной. Реакция между молекулами означает выполнение композиции между функциями, в результате чего образуются новые молекулы.

Наиболее известными из моделей искусственной химии являются модели Tierra [29] и Avida [30, 31]. Эти модели представляют собой клеточные пространства, клетки которых занимают цифровые организмы. Цифровые организмы — это программы на ассемблероподобном языке виртуальной ЭВМ. Вся ассемблероподобная программа рассматривается как аналог хромосомы в генетических алгоритмах, в Avida ее называют геномом. В дальнейшем для подобных хромосомных структур будет использоваться термин "цифровая ДНК".

Цифровые организмы в Tierra и Avida борются за модельное пространство и процессорное время, увеличивая численность своего вида путем самокопирования. В модели Avida цифровые организмы также способны обучаться вычислению несложных логических функций. За вычисление требуемых функций цифровой организм получает бонусы, которые означают дополнительные кванты процессорного времени, доступные организму для выполнения.

Среди особенностей Tierra и Avida, существенных с точки зрения практического прикладного программирования, можно отметить следующие:

- самопроизвольного "возникновения жизни" ни в одной из моделей не происходит, для начала эволюции всегда необходим организм-первопредок;
- цифровые организмы в моделях обучаются только вычислению несложных булевых функций, вырастить программу, обладающую действительно полезными свойствами, не удалось, так как для этого необходимо слишком много поколений.

Таким образом, модели Tierra и Avida представляют больше академический, чем практический интерес.

Неравновесные программные системы

В отличие от цифровых организмов моделей Tierra и Avida, программы, созданные в рамках концепции неравновесного программирования, могут иметь практическую ценность. Концепция неравновесного программирования была предложена автором в работе [32]. В основу была положена аналогия между выполнением программы и химическими реакциями.

В первую очередь предполагается, что самоорганизующаяся программа представляет собой системный объект с избыточным количеством элементов — цифровых организмов. Как самоорганизующаяся система, программная система совершает работу против равновесия за счет имеющейся свободной энергии. Равновесие можно понимать как в химическом, так и в термодинамическом смысле. В первом случае равновесие означает равновесие между прямыми и обратными реакциями, которое можно рассматривать как отсутствие реакций. Во втором случае равновесие означает отсутствие в системе свободной энергии и максимальную энтропию. И в том и в другом случаях для программной системы равновесие означает, что она перестала выполняться. В этом случае количество доступных программной системе квантов процессорного времени (аналогов свободной энергии) равно нулю, а структуры, представляющие программную систему в памяти, подчиняются принципу организации, они статичны и постепенно разрушаются, так как соответствующие области памяти затираются структурами, представляющими другие программы. Так постепенно увеличивается энтропия участка памяти, ранее занимаемого программой.

Таким образом, совершение работы против равновесия означает, что программная система стремится выполняться как можно дольше, высвобождая для этого "энергию", аналогом которой является процессорное время. Свободная энергия в свою очередь определяется

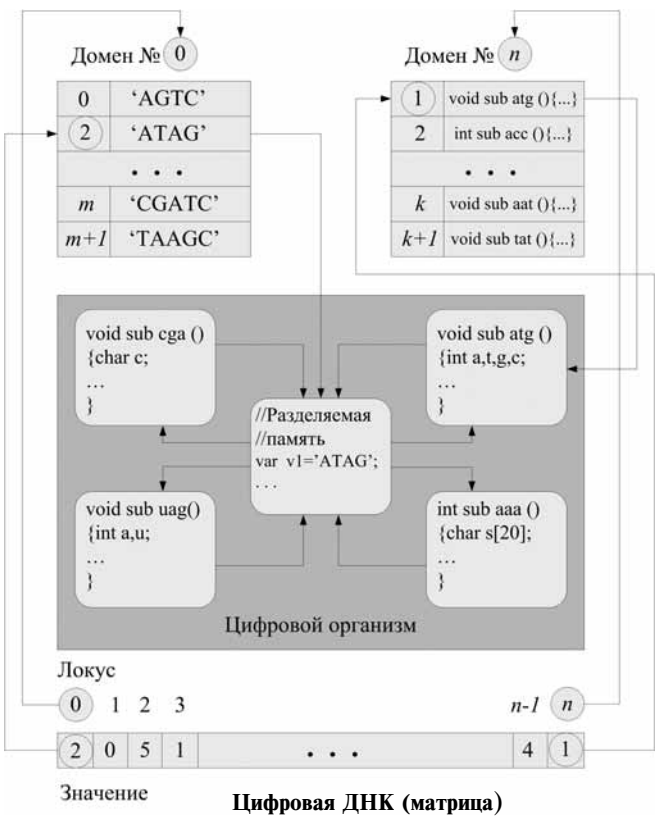
числом баллов или бонусов, полученных всей программной системой и отдельными цифровыми организмами за выполнение полезных с точки зрения пользователя задач. В предельном случае программная система может даже перенастраиваться на выполнение других задач или отыскивать новые задачи, для выполнения которых образуется дочерняя программная система.

Следовательно, целью существования программной системы является как можно более долгое нахождение в неравновесном состоянии, т. е. как можно более долгое выполнение. При этом, решая сугубо эгоистическую задачу, программная система способствует удовлетворению потребностей пользователя.

Предполагается, что программная система обладает самоорганизацией континуального типа, т. е. представляет собой кинетический континуум относительно независимых программных единиц и взаимодействий между ними. Каждая такая единица является цифровым организмом, также представляющим собой неравновесный объект, или кинетический континуум взаимодействующих элементов — программных компонентов и глобальных переменных.

Взаимодействия между элементами по своей сути аналогичны типам химических реакций, происходящих в живых организмах произвольной степени сложности, таких как:

- матричные реакции синтеза (сборка цифрового организма на основе цифровой ДНК и библиотеки компонентов);



Цифровая ДНК (матрица)

- автокаталитические реакции (воспроизводство цифровым организмом потомков путем клонирования цифровой ДНК как точного, так и с ошибкой);
- сопряженные с автокатализом реакции, высвобождающие энергию (реализация основной прикладной целевой функции, выполняемой цифровым организмом в составе программной системы).

Цифровой организм представляет собой результат сборки на основе цифровой ДНК и библиотеки (см. рисунок [24]). Каждый участок цифровой ДНК (локус) имеет жестко заданную семантику и соответствует либо макрооперации, или поведенческому блоку, в составе алгоритма поведения цифрового организма, либо одной из глобальных переменных. Значение локуса, задаваемое целым числом, определяет или конкретную программную реализацию макрооперации в составе алгоритма поведения, или значение переменной. Номер локуса задает номер домена программных реализаций или значений переменной, а значение локуса является номером значения в домене.

Пересборка цифрового организма проводится периодически после каждого полного цикла выполнения алгоритма поведения. Благодаря этому могут быть учтены изменения, произошедшие с организмом вследствие мутации цифровой ДНК. Каждый раз сборка сопровождается анализом цифровой ДНК в целях определения допустимости существования цифрового организма с заданными ДНК свойствами на основании базы правил. Организмы с недопустимыми свойствами или их сочетаниями уничтожаются. Таким образом, свойства программной системы контролируемы и управляемы. Как и в случаях Tierra и Avida, исполнение программ должно осуществляться в среде виртуальной ЭВМ.

Способность подобных программных систем к самоорганизации была подтверждена экспериментально [33].

Предложенная концепция позволяет приблизить процесс создания программных систем к концепции эволюционной эмбриологии [34], в соответствии с которой техническая система может рассматриваться как результат развития из некоторой зародышевой структуры. В работах [35, 36] описан процесс генерации многовидовых иерархий эволюционирующих программ различного назначения на основе исходной популяции, состоявшей из двух цифровых организмов.

Заключение

Внедрение самоорганизующихся программных систем может оказаться перспективой не столь далекого будущего. Областями применения подобных систем в первую очередь могут стать автономные системы администрирования распределенных вычислительных систем и сетей, SaaS (Software as a Service) системы.

Как отмечалось ранее, возможно использование самоорганизации на отдельных этапах жизненного цикла ПО, а именно при его разработке, с последующей фиксацией полученных программных решений.

Еще одной сферой применения самоорганизующихся программных систем может стать моделирование экологических и социальных процессов.

Широкое внедрение самоорганизующихся программных систем потребует разработки новых методов и технологий проектирования и реализации подобных систем, основанных на моделях математической биологии. Во многом может измениться и сфера деятельности программиста. Наряду с разработкой традиционного детерминированного программного обеспечения появится новое направление, связанное с проектированием программных систем как иерархических сообществ, функционирующих на границе порядка и хаоса и развивающихся из первичной зародышевой структуры.

Список литературы

1. **Кольчугина Е. А.** Особенности современных технологий разработки программного обеспечения // Новые информационные технологии и системы: Труды VII Международной научно-технической конференции. Пенза, ПГУ, 2006. Ч. 2. С. 129—132.

2. **Вендров А. М.** CASE-технологии. Современные методы и средства проектирования информационных систем. М.: Финансы и статистика, 1998. 176 с.

3. **Буч Г.** Объектно-ориентированный анализ и проектирование с примерами приложений на C++. 2-е изд. Пер. с англ. М.: Бином, СПб.: Невский диалект, 1999 г. 560 с.

4. **Чарнецки К., Айзенекер У.** Порождающее программирование: методы, инструменты, применение. Для профессионалов. СПб.: Питер, 2005. 731 с.

5. **Гринфилд Дж., Шорт К., Кук С., Кент С., Крупи Дж.** Фабрики разработки программ: потоковая сборка типовых приложений, моделирование, структуры и инструменты. Пер. с англ. М.: Вильямс, 2007. 592 с.

6. **Рассел С., Норвиг П.** Искусственный интеллект: современный подход, 2-е изд. Пер. с англ. М.: Вильямс, 2006. 1408 с.

7. **Воеводин В. В., Воеводин Вл. В.** Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.

8. **Гофф М. К.** Сетевые распределенные вычисления: достижения и проблемы. Пер. с англ. М.: КУДИЦ-ОБРАЗ, 2005. 320 с.

9. **Васенин В. А., Водомеров А. Н., Конев И. М., Степанов Е. А.** Т-подход к автоматизированному распараллеливанию программ: идеи, решения, перспективы / под ред. академика РАН В. А. Садовничего. М.: МЦНМО, 2008. 460 с.

10. **Ратшиллер Т., Геркен Т.** PHP4: разработка Web-приложений. Библиотека программиста. СПб: Питер, 2001. 384 с.

11. **Фогель Л., Оуэнс А., Уолш М.** Искусственный интеллект и эволюционное моделирование: Пер. с англ. М.: Мир, 1969. 232 с.

12. **Koza J. R., Bennett F. H, Andre D., Keane M. A.** Genetic Programming: Biologically Inspired Computation that Creatively Solves Non-Trivial Problems // Evolution as Computation. DIMACS Workshop, Princeton. January 1999. Heidelberg: Springer-Verlag, 2001. P. 15—44.

13. **Гладков Л. А., Курейчик В. В., Курейчик В. М.** Генетические алгоритмы / под ред. В. М. Курейчика. 2-е изд., испр. и доп. М.: ФИЗМАТЛИТ, 2006. 320 с.

14. **Corno F., Sanchez E., Squillero G.** On the Evolution of Corewar Warriors // CEC2004, Congress on Evolutionary Computation, Portland (Oregon). 20—23 June. 2004. P. 2365—2371.

15. **Pawlak R., Cuesta C. E., Younessi H.** Recombinant Programming. URL: http://hal.archives-ouvertes.fr/docs/00/07/06/23/PDF/RR-5380.pdf

16. **Паун К., Розенберг Г., Саломаа А.** ДНК-компьютер. Новая парадигма вычислений. Пер. с англ. М.: Мир, 2004. 528 с.

17. **Хакен Г.** Информация и самоорганизация. Макроскопический подход к сложным системам: Пер. с англ. / Предисл. Ю. Л. Климонтовича. Изд 2-е, доп. М.: КомКнига, 2005. 248 с.

18. **Пригожин И., Николис Г.** Познание сложного: Введение. Пер. с англ. / Предисл. Г. Г. Малинецкого. Изд. 3-е, доп. М.: Издательство ЛКИ, 2008. 352 с.

19. **Гленсдорф П., Пригожин И.** Термодинамическая теория структуры, устойчивости и флуктуаций. Пер. с англ. / Предисл. Г. Г. Малинецкого. Под ред. Ю. А. Чизмарджиева. Изд. 2-е. М.: Едиториал УРСС, 2003. 280 с.

20. **Эбелинг В.** Образование структур при необратимых процессах: Введение в теорию диссипативных структур. М.-Ижевск: Институт компьютерных исследований, НИЦ "Регулярная и хаотическая динамика", 2004. 256 с.

21. **Эбелинг В., Энгель А., Файстель Р.** Физика процессов эволюции. Пер. с нем. Ю. А. Данилова. М.: Эдиториал УРСС, 2001. 328 с.

22. **Галимов Э. М.** Феномен жизни: Между равновесием и нелинейностью. Происхождение и принципы эволюции. Изд. 3-е. М.: ЛИБРОКОМ, 2009. 256 с.

23. **Руденко А. П.** Самоорганизация и синергетика. URL: http://spkurdyumov.narod.ru/rudenko1.htm

24. **Кольчугина Е. А.** Структура цифрового организма в самоорганизующихся программных системах // Программные продукты и системы. 2012. № 2. С. 51—54.

25. **Langton C. G.** Artificial Life // Artificial Life. Santa Fe Institiute Studies in Science of Complexity. Reading, MA: Addison-Wesley, 1988. Vol. IV. P. 1—47.

26. **Эвери Д.** Теория информации и эволюция. М.-Ижевск: Институт компьютерных исследований, НИЦ "Регулярная и хаотическая динамика", 2006. 252 с.

27. **Dittrich P., Ziegler J., Banzhaf W.** Artificial Chemistries — A Review // Artificial Life. 2001. Vol. 7 (N 3). P. 225—275.

28. **Fontana W.** Algorithmic Chemistry // Artificial Life II. Santa fe Institiute Studies in the Sciences of Complexity / Eds. C. G. Langton, C. Taylor, J. D. Farmer, S. Rasmussen. Redwood City, CA: Addison-Wesley. 1991. Vol. X. P. 159—209.

29. **Thearling K., Ray T. S.** Evolving Multi-Cellular Artificial Life// Artificial Life IV. Proceedings of the Forth International Workshop on Synthesis and Simulation of Living Systems / Eds. R. A. Brooks, P. Maes. Cambrige, MA: MIT Press, 1994. P. 283—288.

30. **Adami C.** On Modelling Life // Artificial Life IV. Proceedings of the Forth International Workshop on Synthesis and Simulation of Living Systems / Eds. R. A. Brooks, P. Maes. Cambrige, MA: MIT Press, 1994. P. 269—276.

31. **Adami C., Brown T.** Evolutionary Learning in the 2d Artificial Life System "Avida" // Artificial Life IV, Proceedings of the Forth International Workshop on Synthesis and Simulation of Living Systems / Eds. R. A. Brooks, P. Maes. Cambrige, MA: MIT Press, 1994. P. 377—381.

32. **Кольчугина Е. А.** Неравновесное программирование // Известия высших учебных заведений. Поволжский регион. Технические науки. 2009. № 3 (11). С. 25—31.

33. **Кольчугина Е. А.** Результаты эксперимента по созданию эволюционирующего программного обеспечения // Известия высших учебных заведений. Поволжский регион. Технические науки. 2007. № 1. С. 54—60.

34. **Stanley K. O., Miikkulainen R.** A Taxonomy for Artificial Embryogeny // Artificial Life. 2003. Vol. 9 (N 2). P. 93—130.

35. **Кольчугина Е. А.** Генерация многовидовых иерархий эволюционирующих программ // Известия высших учебных заведений. Поволжский регион. Технические науки. 2007. № 2. С. 48—55.

36. **Кольчугина Е. А.** Новые парадигмы распределенных вычислений и неравновесное программирование // Известия высших учебных заведений. Поволжский регион. Технические науки. 2009. № 4 (12). С. 3—11.

УДК 519.256

А. А. Адуенко, студент, Московский физико-технический институт, e-mail: aduenko1@gmail.com,
В. В. Стрижов, канд. физ.-мат. наук, науч. сотр., Вычислительный центр им. А. А. Дородницына РАН, г. Москва, e-mail: strijov@ccas.ru

Алгоритм оптимального расположения названий коллекции документов[1]

Исследуется метод визуализации результатов тематической кластеризации коллекции документов. Матрица парных расстояний между документами оптимальным способом спроецирована на плоскость. Требуется расположить названия документов оптимальным образом. Предложена такая функция потерь, которая позволяет расположить название тем на плоскости с минимальным перекрытием. Для ее минимизации использован алгоритм BFGS. Алгоритм проиллюстрирован примером визуализации тезисов конференции.

Ключевые слова: *визуализация, тематическая классификация, коллекция документов, функция потерь, алгоритм BFGS*

Введение

Эффективным способом анализа коллекции документов является визуализация матрицы парных расстояний между документами. Каждый документ представлен точкой с подписью. Требуется спроецировать множество точек на плоскость с минимальной, в терминах некоторого функционала, потерей информации [1] и сделать подписи к спроецированным точкам.

Существуют прикладные программы, которые применяют при решении подобных задач. Например, для визуализации графов используется программа Graphviz (graphviz.org). Однако, несмотря на свою распространненость, эта программа визуализирует только граф связей между вершинами. Иначе говоря, на вход программы подаются только ребра между вершинами графа, а фиксировать координаты части вершин графа нельзя. По этой причине для решения рассматриваемой задачи подобные программы неприменимы.

Для визуализации коллекции текстовых документов вводится функция расстояния [2] на множестве документов. Рассмотрим словарь — множество слов, каждое из которых хотя бы раз встретилось в одном из документов коллекции. В данной работе под документом будем понимать неупорядоченное множество слов из словаря. Слова в документе могут повторяться: документ представлен в виде "мешка слов" [3]. Каждому документу поставим в соответствие вектор, содержащий информацию о словарном составе документа. Размерность этого вектора равна числу слов в словаре. Расстояние между документами есть расстояние между векторами, соответствующими этим документам [4, 5].

Для проецирования описывающих документы коллекции векторов в двумерное пространство используется метод главных компонент [1]. Для определения оптимального положения подписей вводится функция потерь. В данной работе ставится задача оптимизации предложенной функции потерь. Она выполняется с помощью метода BFGS (англ. *Broyden—*

[1] Работа выполнена при поддержке Министерства образования и науки РФ в рамках Государственного контракта 07.524.11.4002.

Fletcher—Goldfarb—Shanno method) [6, 7]. В качестве начального приближения используется случайное приближение вблизи исходных точек. Результаты оптимизации функций сравниваются.

Предложенный алгоритм проиллюстрирован вычислительным экспериментом на данных конференции "European Conference on Operational Research, EURO-2012". В качестве коллекции документов были взяты 48 тезисов докладов конференции из раздела "DEA and performance measurement".

Постановка задачи

Пусть $W = \{w_1, ..., w_n\}$ — заданное множество слов, словарь. Документом назовем неупорядоченное множество слов из W: $\{w_j\}$, $w_j \in W$ — слово в документе. Представим документ с номером i в виде вектора

$$\mathbf{x}_i = [c(i, w_1), ..., c(i, w_j), ..., c(i, w_n)]^{\mathrm{T}},$$

где $c(i, w_j)$ — число вхождений слова w_j в i-й документ. Опишем множество документов следующим образом. Задана выборка — множество n векторов признаков документов $\mathbf{x} \in \Re^n$

$$\{\mathbf{x}_i\},\ i \in \{1, ..., m\},$$

где m — число документов в коллекции, а $\Re$ — множество неотрицательных действительных чисел.

Задана функция расстояния между векторами признаков документов — евклидова метрика

$$\rho(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{j=1}^{n} \left|x_j - x'_j\right|^2},$$

где x_j — компонента вектора $\mathbf{x}$. Построим матрицу попарных расстояний между всеми парами векторов $\mathbf{x}_i$, $\mathbf{x}_k$, $i, k \in \{1, ..., m\}$. Для визуализации взаимного расположения документов, описанного матрицей попарных расстояний, предлагается найти ее оптимальную проекцию на плоскость.

Документ представлен точкой на плоскости, а заголовок документа — текстом с выноской — с линией, соединяющей точку и текст. Оптимальная проекция находится методом главных компонент. В случае если размерность пространства проекции m не больше ранга матрицы $\mathbf{X}$, матрица проекций $\mathbf{Z}$ находится как $\mathbf{Z} = \mathbf{XU}$. Здесь $\mathbf{U}$ — ортонормированная матрица, $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}_m$, где $\mathbf{I}_m$ — единичная матрица размерности m. Матрица $\mathbf{U}$ состоит из собственных векторов матрицы $\mathbf{X}^{\mathrm{T}}\mathbf{X}$, соответствующих m максимальным собственным числам $\lambda_1, ..., \lambda_m$ этой матрицы [1]

Для визуализации подписей к документам поставим в соответствие каждому вектору $\mathbf{z}$, состоящему из двух компонент и являющемуся строкой матрицы проекций $\mathbf{Z}$, вектор $\mathbf{t} \in \Re^4$ с координатами, задающими прямоугольник на плоскости $\mathbf{t} = [t_1, ..., t_4]$. Здесь

t_1 — координата нижнего левого угла по оси абсцисс;

t_2 — координата нижнего левого угла по оси ординат;

t_3 — ширина текста;

t_4 — высота текста.

При этом допускается отрицательное значение ширины. Оно будет указывать на то, что линия выноски соединяет точку с правым нижним углом. Если ширина текста $t_3 \geqslant 0$, то линия выноски соединяет точку с левым нижним углом. Значения $|t_3|$ и t_4 вычисляются исходя из длины текста и размера шрифта.

Введем функцию потерь $S(\mathbf{T}|\mathbf{Z})$, определенную на матрице $\mathbf{T} = \left[\mathbf{t}_1^{\mathrm{T}}, ..., \mathbf{t}_m^{\mathrm{T}}\right]^{\mathrm{T}}$, при заданной матрице $\mathbf{Z}$. Оптимальная матрица $\hat{\mathbf{T}}$ будет выбираться из условия минимума $S(\mathbf{T}|\mathbf{Z})$:

$$\hat{\mathbf{T}} = \arg \min_{\mathbf{T} \in \Re_+^{m \times 4}} S(\mathbf{T}|\mathbf{Z}).$$

Построение функции потерь

Для построения функции потерь S рассмотрим объекты трех типов: прямоугольник, соответствующий визуализируемому тексту; отрезок, соединяющий точку и текст; точку, обозначающую документ. Функция потерь $S(\mathbf{T}|\mathbf{Z})$, зависящая от координат прямоугольников $\mathbf{T}$ при заданных координатах точек $\mathbf{Z}$, имеет вид

$$S(\mathbf{T}|\mathbf{Z}) = \frac{1}{m(m-1)} \sum_{i \neq k} \sum [w_\alpha \alpha(\mathbf{z}_i, \mathbf{z}_k, \mathbf{t}_k) + w_\delta \delta(\mathbf{z}_i, \mathbf{t}_i, \mathbf{t}_k)] +$$
$$+ \frac{2}{m(m-1)} \sum_{i > k} \sum [w_\beta \beta(\mathbf{z}_i, \mathbf{t}_i, \mathbf{z}_k, \mathbf{t}_k) + w_\eta \eta(\mathbf{t}_i, \mathbf{t}_k)] +$$
$$+ \frac{1}{m^2} \sum_{i,\,k} [w_\gamma \gamma(\mathbf{z}_i, \mathbf{t}_k)] + \frac{1}{m} \sum_i [w_\rho \rho(\mathbf{t}_i, \mathbf{z}_i)], \qquad (1)$$

где $i, k \in \{1, ..., m\}$.

В выражении (1) множители перед суммами введены для нормировки соответствующих штрафов. Веса штрафов w_α, w_β, w_γ, w_δ, w_η, w_ρ введены для управления вкладом каждого штрафа в функции потерь S. Опишем каждое слагаемое в функции (1) в отдельности. Слагаемое ρ штрафует за удаление надписи на большое расстояние от соответствующей точки.

$$\rho(\mathbf{t}_i, \mathbf{z}_k) = \|\mathbf{z}_k - \tilde{\mathbf{t}}_i\|_2,$$

где $\tilde{\mathbf{t}}_i$ — вектор, содержащий координаты точки прямоугольника $\mathbf{t}_i$, в которой кончается линия выноски, $\tilde{\mathbf{t}} = [t_1, t_2]$. Это, в соответствии с ГОСТ 2.316—68 [8], может быть нижний левый или нижний правый угол прямоугольника.

Штрафы за наложение объектов

Объекты, на которые происходит наложение	Объекты, которые накладываются		
	Точка	Линия	Прямоугольник
Точка	—	—	—
Линия	α	β	—
Прямоугольник	γ	δ	η

Остальные пять функций α, β, γ, δ, η отражают штрафы за наложение одного объекта на другой. В таблице указано, что обозначает каждая функция.

В строках таблицы указаны объекты, на которые происходит наложение. В столбцах таблицы указаны объекты, которые накладываются. Прочерк в ячейке таблицы означает, что данное наложение не штрафуется функцией потерь (1). Определим оставшиеся пять слагаемых в функции потерь $S(\mathbf{T}|\mathbf{Z})$.

Для этого определим расстояние l от некоторой точки $\mathbf{z}_i$ до выноски $(\mathbf{z}_k, \tilde{\mathbf{t}}_k)$ как евклидово расстояние от точки $\mathbf{z}_i$ до ее проекции $\mathbf{p}_i$ на этот отрезок:

$$l = \|\mathbf{x}_i - \mathbf{p}_i\|_2.$$

Запишем $\mathbf{p}_i$ в виде

$$\mathbf{p}_i = \zeta \mathbf{x}_k + (1 - \zeta)\tilde{\mathbf{t}}_k. \quad (2)$$

Из свойств проекции получаем, что

$$\zeta = \frac{(z_i - \tilde{\mathbf{t}}_k)^{\mathrm{T}}(\mathbf{z}_k - \tilde{\mathbf{t}}_k)}{\|\mathbf{z}_k - \tilde{\mathbf{t}}_k\|^2}. \quad (3)$$

Значение вектора $\mathbf{p}_i$ находится по известной ζ с помощью формулы (2).

Слагаемое-штраф $\alpha(\mathbf{z}_i, \mathbf{z}_k, \mathbf{t}_k)$ зададим в виде

$$\alpha(\mathbf{z}_i, \mathbf{z}_k, \mathbf{t}_k) = g\max\left(\frac{h - l}{h}, 0\right),$$

где h — ширина прямоугольника, а множитель

$$g = \begin{cases} 1, \text{ если } \zeta \in [0, 1], \\ 0 \text{ — в противном случае,} \end{cases} \quad (4)$$

где ζ вычисляется по формуле (3). Однако определенная формулой (4) g, а значит, и α не являются непрерывными по ζ, что затрудняет оптимизацию α. По этой причине в качестве g рассмотрим

$$\mathrm{g} = \begin{cases} \zeta(1 - \zeta), \zeta \in [0, 1], \\ 0, \zeta \notin [0, 1]. \end{cases}$$

Определим штраф $\beta(\mathbf{z}_i, \mathbf{t}_i, \mathbf{z}_k, \mathbf{t}_k)$ за пересечение отрезков выносок $(\mathbf{z}_i, \tilde{\mathbf{t}}_i)$ и $(\mathbf{z}_k, \tilde{\mathbf{t}}_k)$. Если отрезки не пересекаются, полагаем $\beta = 0$, иначе находим точку $\mathbf{z}$ пересечения отрезков. Записываем ее в виде

$$\mathbf{z} = \zeta_1 \mathbf{z}_i + (1 - \zeta_1)\tilde{\mathbf{t}}_i = \zeta_2 \mathbf{z}_k + (1 - \zeta_2)\tilde{\mathbf{t}}_k.$$

Заметим, что когда отрезки $(\mathbf{z}_i, \tilde{\mathbf{t}}_i)$ и $(\mathbf{z}_k, \tilde{\mathbf{t}}_k)$ пересекаются, то $\mathbf{z}$ находится внутри каждого из них и ζ_1, $\zeta_2 \in [0, 1]$. Ситуация $\zeta_1 = 0; 1$ или $\zeta_2 = 0; 1$ соответствует касанию отрезков. Укажем, как найти точку пересечения отрезков. Для этого найдем точку пересечения прямых, содержащих эти отрезки, если таковая существует. Заметим, что в данной задаче исходные точки разные, потому разные отрезки не могут полностью совпадать. Запишем условие пересечения прямых

$$\mathbf{z}_i + \zeta_1(\tilde{\mathbf{t}}_i - \mathbf{z}_i) = \mathbf{z}_k + \zeta_2(\tilde{\mathbf{t}}_k - \mathbf{z}_k)$$

для некоторых $\zeta_1, \zeta_2 \in \Re$. Перепишем это условие в виде

$$\zeta_1(\tilde{\mathbf{t}}_i - \mathbf{z}_i) - \zeta_2(\tilde{\mathbf{t}}_k - \mathbf{z}_k) = \mathbf{z}_k - \mathbf{z}_i.$$

Последнее выражение есть система линейных уравнений на ζ_1, ζ_2

$$\mathbf{A}\zeta = \mathbf{b},$$

где $\mathbf{A} = (\tilde{\mathbf{t}}_i - \mathbf{z}_i, -\tilde{\mathbf{t}}_k + \mathbf{z}_k)$, $\mathbf{b} = \mathbf{z}_k - \mathbf{z}_i$. Если матрица $\mathbf{A}$ вырождена, т. е. $\det(\mathbf{A}) = 0$, то отрезки параллельны. Иначе можно однозначно найти $\zeta = \mathbf{A}^{-1}\mathbf{b}$. В этом случае прямые, соответствующие отрезкам, пересекаются. Тогда и только тогда, когда пересекаются сами отрезки, выполнено $\zeta_1, \zeta_2 \in [0, 1]$.

Определим функцию $\varphi(\xi)$, $\xi \in \Re$:

$$\varphi(\xi) = \begin{cases} \xi, \text{ если } \xi > 0 \\ 0, \text{ если } \xi \leqslant 0. \end{cases}$$

С помощью φ штраф за пересечение отрезков $\beta(\mathbf{z}_i, \mathbf{t}_i, \mathbf{z}_k, \mathbf{t}_k)$ можно определить как

$$\beta(\mathbf{z}_i, \mathbf{t}_i, \mathbf{z}_k, \mathbf{t}_k) = \varphi(\zeta_1(1 - \zeta_1))\varphi(\zeta_2(1 - \zeta_2)).$$

Таким образом, штраф $\beta(\mathbf{z}_i, \mathbf{t}_i, \mathbf{x}_k, \mathbf{t}_k)$ будет обладать непрерывностью по $\mathbf{t}_i$, $\mathbf{t}_k$.

Определим штраф $\gamma(\mathbf{z}_i, \mathbf{t}_k)$ за попадание точки $\mathbf{z}_i$ внутрь прямоугольника $\mathbf{t}_k$. Обозначая $d_k = t_{k3}$ ширину прямоугольника, а $h = t_{k4}$ высоту, определим ξ_1, ξ_2 следующим образом:

$$\xi_1 = \frac{|z_{i1} - \hat{t}_{k1}|}{\frac{d_k}{2}}, \quad \xi_2 = \frac{|z_{i2} - \hat{t}_{k2}|}{\frac{h}{2}},$$

где $\hat{t}_{k1}$ и $\hat{t}_{k2}$ — координаты центров прямоугольников по осям абсцисс и ординат соответственно:

$$\hat{t}_{k1} = t_{k1} + \frac{1}{2}t_{k3},\ \hat{t}_{k2} = t_{k2} + \frac{1}{2}t_{k4}.$$

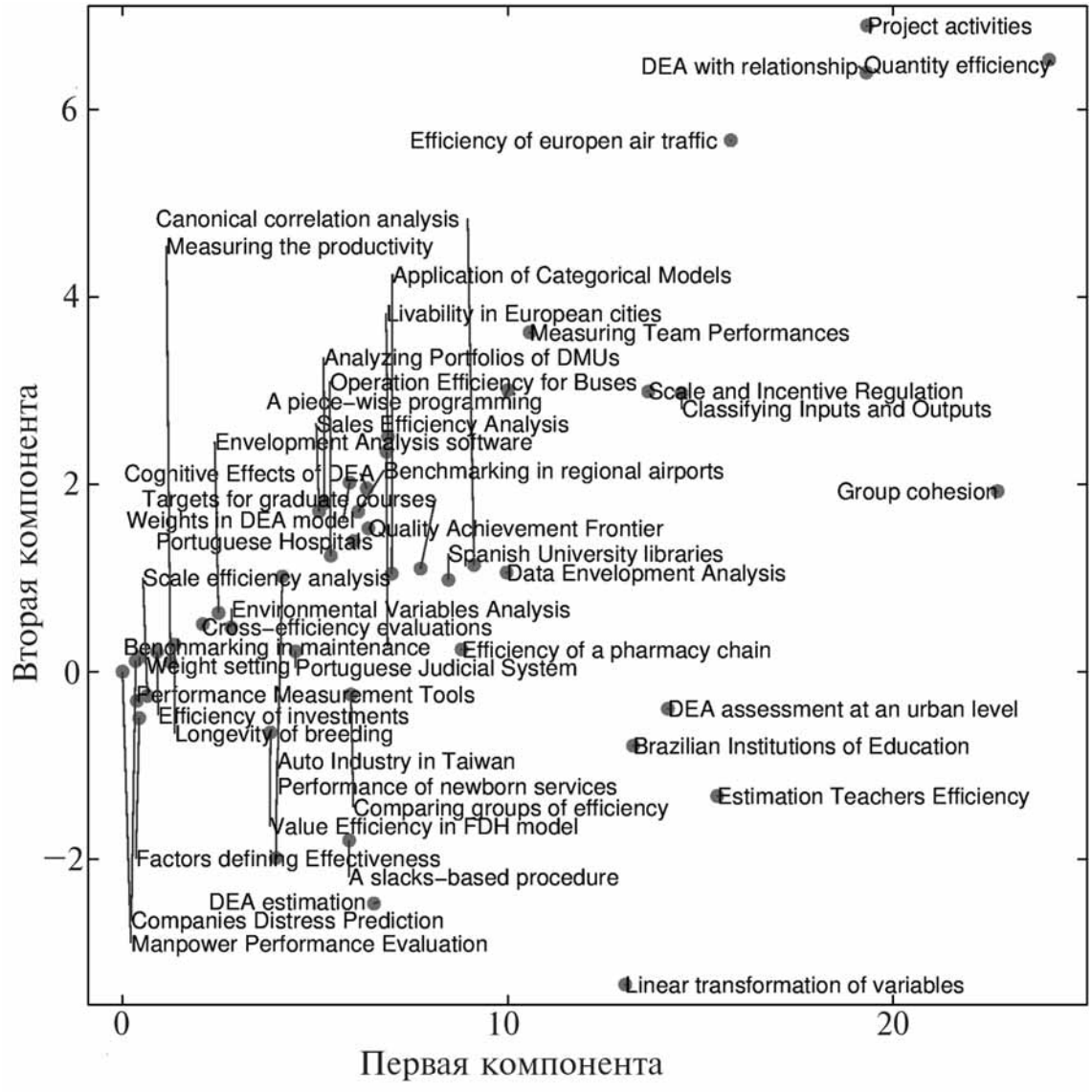
В качестве штрафа $\gamma(\mathbf{z}_i, \mathbf{t}_k)$ возьмем

$$\gamma(\mathbf{z}_i, \mathbf{t}_k) = \varphi(1 - \xi_1)\varphi(1 - \xi_2).$$

Этот штраф тоже непрерывен в силу непрерывности φ и непрерывной зависимости ξ_1 и ξ_2 от $\mathbf{z}_i$ и $\mathbf{t}_k$.

Определим штраф δ за пересечение отрезка выноски и прямоугольника. Если отрезок $(\mathbf{z}_i, \tilde{\mathbf{t}}_i)$ и прямоугольник $\mathbf{t}_k$ не пересекаются, то $\delta(\mathbf{z}_i, \mathbf{t}_i, \mathbf{t}_k) = 0$. Если пересечение имеет место, рассмотрим две фигуры G_0 и G_1, на которые разбивается прямоугольник прямой p, содержащей отрезок пересечения. Вычислим площади S_0 и S_1 каждой из них и поделим меньшую $S_{\min}$ на большую $S_{\max}$: $f = \frac{S_{\min}}{S_{\max}}$. Подсчитаем также общую длину l отрезка, находящуюся внутри прямоугольника, и полную длину L отрезка, соединяющего точки пересечения прямой p с границей прямоугольника. Определим штраф $\delta(\mathbf{z}_i, \mathbf{t}_k)$ следующим образом:

$$\delta(\mathbf{z}_i, \mathbf{t}_k) = \frac{l}{L} f.$$



Результат оптимизации функции потерь при
$w_\alpha = w_\beta = w_\delta = 0$, $w_\rho = 10$, $w_\eta = 10000$, $w_W = 10$, $w_H = 10$, $w_\gamma = 1000$

Определим штраф $\eta(\mathbf{t}_i, \mathbf{t}_k)$ за пересечение прямоугольников. Вычислим сначала площадь пересечения прямоугольников $\mathbf{t}_i$, $\mathbf{t}_k$. Заметим, что при пересечении прямоугольников, стороны которых параллельны осям координат, получается также прямоугольник, стороны которого параллельны осям координат. Осталось только определить координаты его левого нижнего и правого верхнего углов. Первая координата левого нижнего угла есть максимум из соответствующих координат пересекающихся прямоугольников $\mathbf{t}_i$, $\mathbf{t}_k$. Вторая компонента — также максимум из соответствующих координат $\mathbf{t}_i$, $\mathbf{t}_k$. Координаты правого верхнего угла равны минимуму соответствующих координат правых верхних углов пересекающихся прямоугольников $\mathbf{t}_i$, $\mathbf{t}_k$. Тогда, добавляя условие существования прямоугольника в площадь пересечения C, получим

$$C = \varphi(\min(t_{i3} + t_{i1}, t_{k3} + t_{k1}) - \max(t_{i1}, t_{k1})) \times \\ \times \varphi(\min(t_{i4} + t_{i2}, t_{k4} + t_{k2}) - \max(t_{i2}, t_{k2})). \quad (5)$$

Нормируя полученную площадь пересечения на площадь меньшего из прямоугольников, получим функцию штрафа $\eta(\mathbf{t}_i, \mathbf{t}_k)$:

$$\eta(\mathbf{t}_i, \mathbf{t}_k) = \frac{C}{\min(S_1, S_2)},$$

где C определяется формулой (5). Построенный штраф также будет непрерывным по $\mathbf{t}_i$, $\mathbf{t}_k$.

Вычислительный эксперимент

В вычислительном эксперименте предложенный алгоритм тестировался на данных 48 тезисов докладов конференции "European Conference on Operational Research, EURO-2012" из раздела "DEA and performance measurement". Минимизация функции потерь (1) проводилась с помощью алгоритма BFGS. Для уменьшения числа шагов минимизации после каждого ее шага применялись разные эвристики. Приведем далее некоторые из них.

- **Стягивание отрезка выноски.** Пытаемся сдвинуть выноску вдоль линии выноски в направлении к исходной точке с некоторым шагом. Стягивание происходит, если значение минимизируемой функции (1) уменьшается по сравнению с исходным.
- **Отражение.** Отражаем выноску относительно вертикальной оси, проходящей через исходную точку, и меняем угол прикрепления выноски. Отражение происходит, если значение минимизируемой функции (1) уменьшается по сравнению с исходным.

Для того чтобы область, где расположены точки и выноски, была как можно ближе к минимально возможной по ширине и высоте, добавим в функцию потерь S (1) штрафы за ширину W и высоту H окна, занимаемого точками. Веса, соответствующие этим штрафам, обозначим w_W и w_H.

Результаты работы алгоритма при заданных значениях весов штрафов, введенных в (1), приведены на рисунке. Площадь пересечения надписей не превосходит 0,1 % от максимальной, а ширина и высота совпадают с шириной и высотой, которые определяются по исходным точкам. Это означает, что ширина и высота минимальны.

Заключение

В данной работе решена задача визуализации коллекции документов в виде точек на плоскости с подписями — названиями документов. Для проектирования векторов, соответствующих документам из коллеции, в плоскость использовался метод главных компонент. Предложена функция потерь для оптимизации расположения подписей. Для нахождения оптимального расположения использовался алгоритм BFGS. В вычислительном эксперименте предложенный алгоритм позволил площадь пересечений выносок свести к нулю, а также оставить размер области визуализации равным минимально возможному. В этой работе описаны алгоритмы, применимые для плоского случая, однако их нетрудно обобщить на случай более высокой размерности.

Список литературы

1. **Jolliffe I. T.** Principle Component Analysis. New York: Springer, 2002.

2. **Blei D. M., Ng A. Y., Jordan M. I.** Latent dirichlet allocation // Journal of Machine Learning Research. 2003. Vol. 3. P. 993—1022.

3. **Hofmann T.** Probabilistic latent semantic indexing // Proceedings of the 22nd annual interanational ACM SIGIR conference on research and development in information retrieval. New York: ACM, 1999. P. 50—57.

4. **Feldman R., Dagan I., Hirsh H.** Mining text using keywords distributions // Journal of Intelligent Information Systems, 1998. Vol. 10 (3). P. 281—300.

5. **Bigi B.** Using Kullback-Leibler distance for text categorization // Advances in information retrieval, 25th Conference on IR research. Berlin: Springer, 2003. P. 305—319.

6. **Liu D. C., Nocedal J.** On the limited memory BFGS method for large scale optimiation // Mathematical programming. 1989. Vol. 45. P. 503—528.

7. **Sahari M. L., Khaldi R.** Quasi-Newton type of diagonal updating for the L-BFGS method // Acta Mathematica Universitatis Comenianae. 2009. Vol. 78. N 2. P. 173—181.

8. **ГОСТ 2.316—68.** Правила нанесения на чертежах надписей, технических требований и таблиц / М.: ИПК Издательство стандартов, 1968.

УДК 002.55, 004.912

Ю. И. Петров, преподаватель, Московский государственный университет приборостроения и информатики,
e-mail: mailtoyuripetrov@gmail.com

Архитектура и алгоритмическое обеспечение информационной системы нормоконтроля выпускных квалификационных работ

Рассматриваются вопросы проектирования и реализации информационной системы нормоконтроля выпускных квалификационных работ. Предложена архитектура программного обеспечения, приведен алгоритм синтаксического анализа документа и пример его реализации на основе объектной модели текстового процессора Microsoft Word.

Ключевые слова: *нормоконтроль, выпускная квалификационная работа, информационная система, программное обеспечение, автоматизация, синтаксический анализ, текстовый процессор, Microsoft Word*

Введение

Настоящая статья развивает положения, опубликованные ранее в работе [1], где отмечалось, что существующее на настоящее время зарубежное программное обеспечение нормоконтроля выпускных квалификационных работ (ВКР) характеризуется ограниченной функциональностью и обладает рядом недостатков. Эти программные разработки слабо адаптированы под российские реалии, их строгая ориентация на международные стандарты и отсутствие локализации не позволяют эффективно использовать данные приложения в отечественной практике. Доступные потенциальным пользователям, немногочисленные российские программные продукты такого назначения не обладают должной функциональностью [2]. Это обстоятельство указывает на актуальность задачи разработки отечественной информационной системы (ИС), позволяющей провести нормоконтроль ВКР — проверку на предмет их соответствия заданным нормам и правилам оформления.

Структура и параметры текста

Понятие ВКР охватывает как дипломный проект (для квалификации специалиста), так и выпускную работу бакалавра, а также магистерскую диссертацию. Общие правила их оформления схожи, поэтому положения настоящей статьи могут быть применены к любому из указанных типов выпускных работ.

Выпускная квалификационная работа, которая подается на нормоконтроль, имеет определенную структуру. Основными составными частями этого документа являются:

- содержание;
- введение;
- разделы;
- заключение;
- список источников.

Все контролируемые области и параметры документа можно подразделить на следующие две категории:

- документ в целом, который включает общие параметры документа, такие как поля, нумерация страниц и подобные им;
- содержимое документа, включающее составные элементы документа и их параметры.

Под составными элементами понимаются отдельные, самостоятельные объекты документа заданного типа, например, текст, рисунок и другие, которые подразумевают собственную обработку и контроль определенных параметров, таких как размер и гарнитура шрифта или междустрочный интервал.

Схема структуры документа и классификация его контролируемых областей и параметров (выделены темно-серым фоном) приведены на рис. 1.

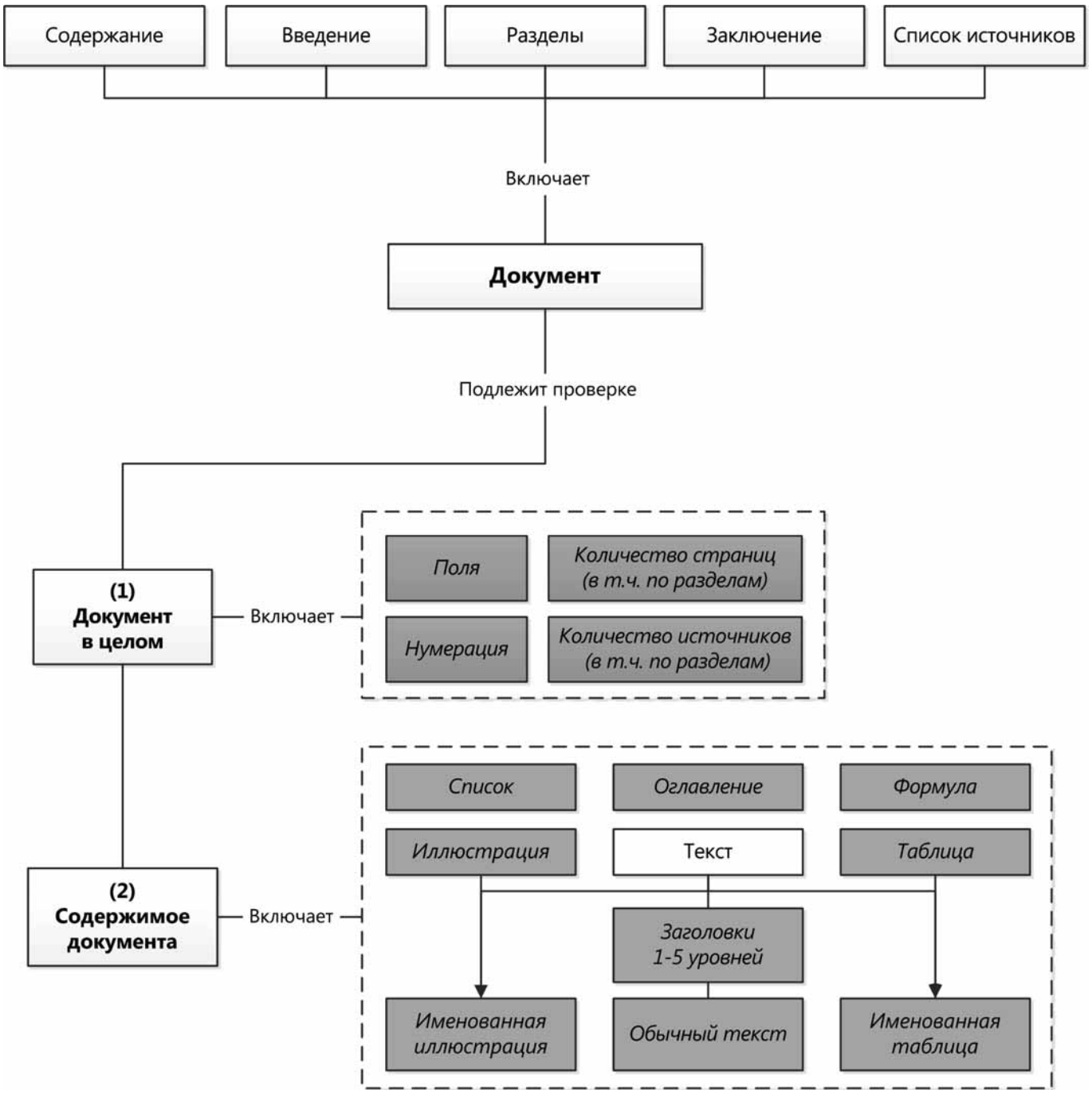


Рис. 1. Общая структура и области проверки документа ВКР

Функциональные и архитектурные особенности ИС

Информационная система нормоконтроля ВКР должна обеспечивать комплексную проверку документа в рамках структуры, представленной на рис. 1. К создаваемой ИС предъявляются следующие функциональные требования.

- Осуществление проверки:
  - общих параметров документа;
  - параметров содержимого документа.
- Поддержка различных режимов проверки (отчет, исправление ошибок или их комбинация).
- Наличие механизмов гибкой настройки системы проверки документов.
- Возможность автоматической генерации отчетов по результатам работы системы.

В качестве пользователя системы рассматривается нормоконтролер, осуществляющий как настройку системы, так и ее эксплуатацию.

Разработка ИС ведется на базе Московского государственного университета приборостроения и информатики, где по окончании проекта также планируется внедрение программного обеспечения в образовательный процесс. В вузе большой парк ПК, при этом большинство настольных компьютеров работает под управлением ОС Windows XP-7. В связи с этим разрабатываемая ИС должна поддерживать данную линейку операционных систем.

На основании предъявляемых к системе требований и учета особенностей среды окружения разработчиками была принята архитектура системы, основные компоненты которой представлены на рис. 2.

Языком реализации бизнес-логики и средой разработки выбран Delphi XE2 — средство для создания

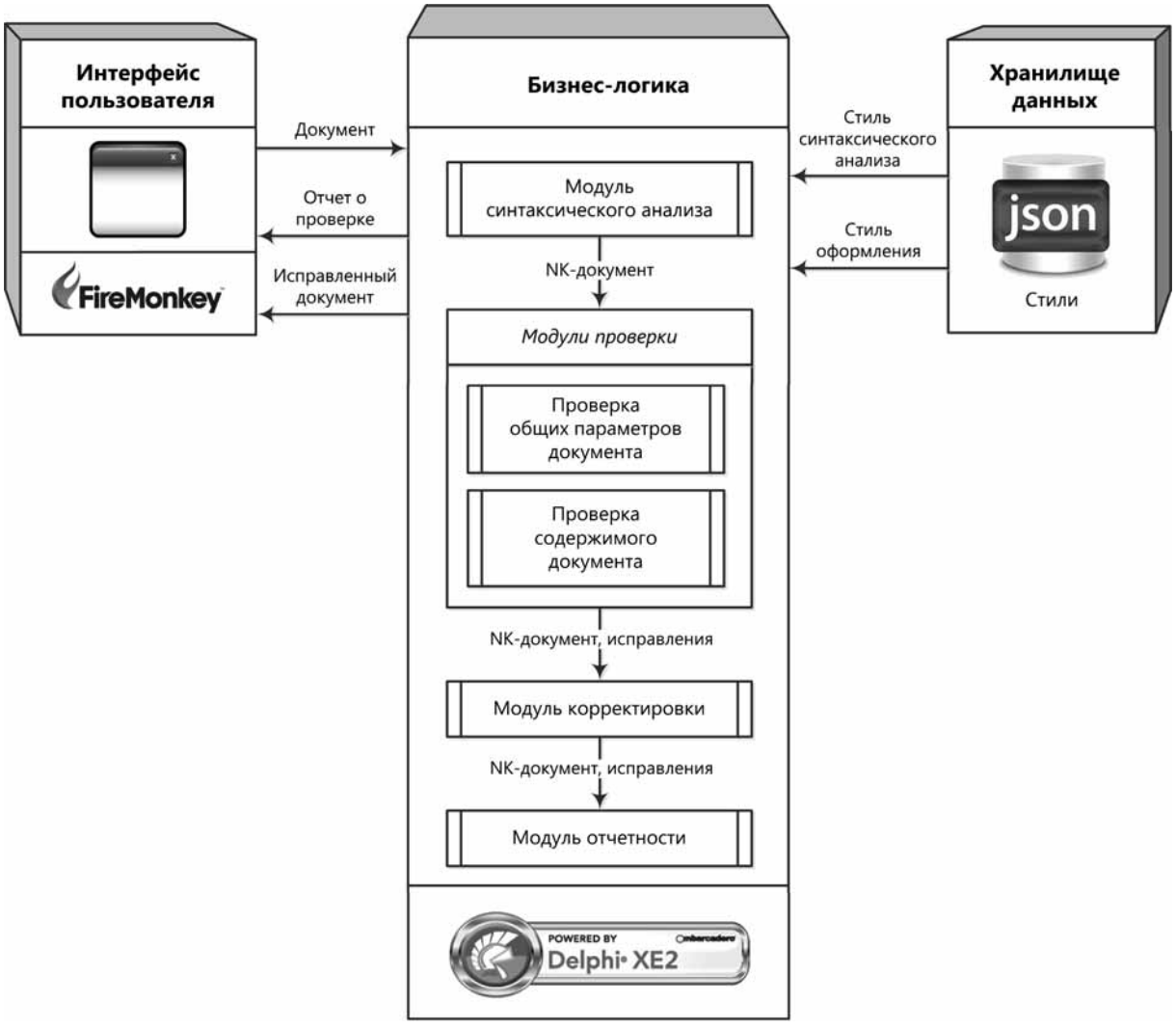


Рис. 2. Общая архитектура ИС нормоконтроля ВКР

платформенно-ориентированных приложений с широкими возможностями для их использования под управлением ОС Windows, Mac и мобильных устройств [3]. В интерфейсной части системы предполагается использовать FireMonkey — библиотеку графического пользовательского интерфейса (GUI), использующую возможности графического процессора [4]. Средства FireMonkey в Delphi позволяют создавать приложения, имеющие высокоинтерактивный привлекательный интерфейс и обладающие разнообразными механизмами обработки данных. Стили, необходимые для работы системы, предполагается хранить в универсальном JSON-формате [5].

Модульная структура системы позволит реализовать ее расширяемое, многофункциональное программное обеспечение [6, 7]. Учитывая, что модули могут разрабатываться независимо друг от друга и иметь свою специфику (например, модуль синтаксического анализа, как будет показано далее), итоговые технические требования к ИС должны быть дополнены соответствующими требованиями к отдельным ее модулям.

Алгоритм проверки документа ВКР

В предыдущем разделе представлена общая архитектура информационной системы нормоконтроля ВКР, а также выделены ее ключевые модули. Реализация системы подразумевает разработку комплексного алгоритма, включающего этапы синтаксического анализа, проверки документа и генерации отчетности.

Модуль синтаксического анализа. Первый этап общего алгоритма, реализующего нормоконтроль ВКР, — выполнение синтаксического анализа документа с текстом выпускной работы. В данном случае под синтаксическим анализом (разбором) понимают процесс выявления структуры документа на основе грамматики языка, на котором он написан. Наиболее известными приложениями, использующими синтаксический разбор, являются поисковые роботы в сети Интернет, трансляторы языков программирования и системы машинного перевода.

Модуль синтаксического анализа в ИС нормоконтроля ВКР отвечает за распознавание в тексте выпускной работы составных элементов и их параметров, а также получение общих контрольных свойств документа. Технически проведение синтаксического разбора возможно с использованием API (application programming interface — интерфейс программирования приложений) текстовых процессоров и вручную. В первом случае основная работа по извлечению информации из документа ложится на текстовый процессор. При этом приложение взаимодействует с редактором посредством API последнего, с помощью набора готовых классов, процедур, функций, структур и констант, которые предоставляются приложением для использования во внешних программных продуктах. Второй вариант подразумевает самостоятельную работу с файлом документа, без использования механизмов (функций) текстовых процессоров. В этом случае также могут быть применены сторонние модули или компоненты (существующие для выбранного языка программирования), поддерживающие работу с файлом документа того или иного типа. Преимущества и недостатки указанных подходов указаны в таблице.

Более подробно вопросы выбора варианта осуществления синтаксического анализа документа обсуждаются в работе [8]. В настоящей статье рассматривается первый вариант с использованием текстового процессора Microsoft Word и технологии OLE Automation. В общем случае документ Microsoft Word является комплексной структурой, содержащей различные типы данных. Для извлечения данных из конкретного документа необходимо провести его синтаксический анализ, а именно — определить типы хранящихся данных и выявить структуру документа [9]. Поддерживая технологию OLE Automation, Word облегчает эту задачу, предоставляя разработчику необходимые данные через объектную модель [10, 11]. Следует отметить, что предлагаемая архитектура ИС нормоконтроля ВКР выделяет операцию синтаксического разбора в отдельный модуль, обеспечивая системе достаточную гибкость для последующего масштабирования. Таким образом, в дальнейшем возможна реализация дополнительных независимых модулей синтаксического анализа. Такой подход позволяет проводить обработку документов, используя как другие офисные пакеты, например, LibreOffice [12] (что, в частности, позволит максимально корректно работать с документами популярного открытого фор-

Преимущества и недостатки подходов к синтаксическому разбору документа

Подход	Преимущества	Недостатки
С использованием API текстовых процессоров	Упрощенное извлечение информации из документа за счет использования предоставляемой оболочки доступа — API текстового процессора	Обязательное наличие (установка) текстового процессора на целевом ПК. Как правило, это приводит к росту минимальных технических требований к компьютеру, на котором будет происходить развертывание, а также увеличению совокупных затрат на приобретение ИС в случае использования коммерческого программного обеспечения (например, Microsoft Word)
Вручную	Отсутствие зависимостей от какого-либо дополнительного программного обеспечения в виде текстовых процессоров	Самостоятельное изучение документации, описывающей формат файла документа, и дальнейший синтаксический анализ содержимого. При отсутствии документации сложность синтаксического разбора многократно возрастает

мата Open Document [13]), так и прочие способы автоматизации, также рассмотренные в работе [8].

Независимо от выбранного способа синтаксического анализа, распознавание составного элемента документа является двухфакторным и базируется на совокупности следующих параметров:

- первый параметр — информация о типе элемента, полученная из документа (с использованием API текстового процессора или вручную);
- второй параметр — информация, полученная из стиля синтаксического анализа для элемента соответствующего типа. Стиль синтаксического анализа содержится в специальном файле (принадлежащем модулю синтаксического анализа) и включает дополнительные правила для распознавания элемента конкретного типа. Фрагмент примера стиля, регламентирующий определение именованной иллюстрации в тексте документа, представлен на Листинге 1.

Листинг 1. Фрагмент примера стиля для модуля синтаксического анализа, регламентирующий определение именованной иллюстрации в тексте документа

```
{
    ...
/* Подпись в именованной иллюстрации должна начинаться с одного из вариантов
массива Prefix, положение подписи (Position) — под рисунком (Below) */

     "PictureWithAnnotation": {
          "Annotation": {
               "Prefix": [
                    "Рисунок",
                    "Рис.",
                    "Иллюстрация",
                    "Image"
               ],
               "Position": "Below"
                ...
          }
     }, ...
}
```

Алгоритм синтаксического анализа документа ВКР приведен на рис. 3.

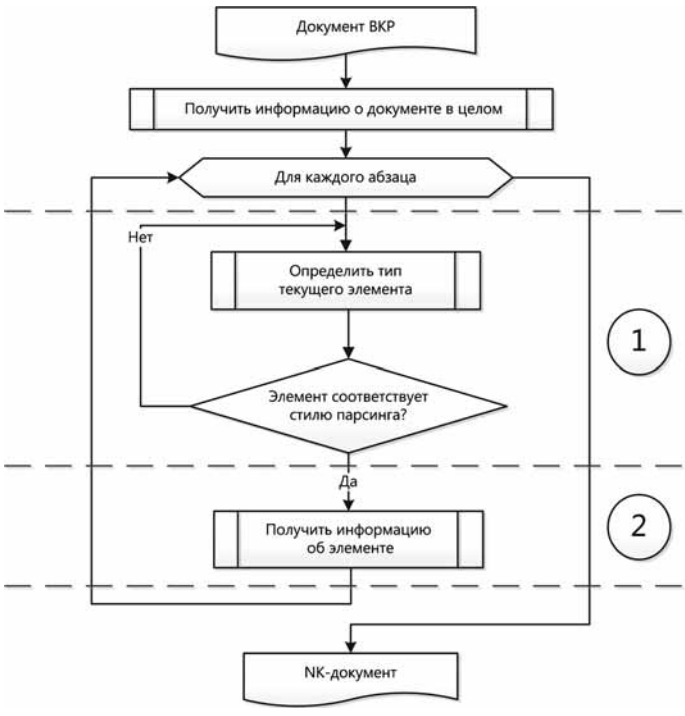


Рис. 3. Алгоритм синтаксического анализа документа ВКР

Модуль последовательно получает общие параметры исходного документа, а затем, перемещаясь по абзацам, определяет составные элементы, их тип и в случае соответствия стилю синтаксического анализа определяет свойства найденных элементов документа.

Наиболее сложная часть алгоритма — определение элемента и его типа (блок 1 на рис. 3). Ключевой особенностью этого блока является порядок определения элементов от оглавления до обычного текста, который описывается следующим логическим выражением:

$$\text{IsToc} \vee \text{IsAnyTable} \vee \text{IsAnyPicture} \vee \vee \text{IsList} \vee \text{IsFormula} \vee \text{IsAnyText} = 1,$$

где IsToc определяет наличие оглавления;

IsAnyTable = IsTableWithAnnotation $\vee$ IsTable определяет наличие таблицы (с заголовком или обычной);

IsAnyPicture = IsPictureWithAnnotation $\vee$ IsPicture определяет наличие иллюстрации (с подписью или обычной);

IsList определяет наличие списка;

IsFormula определяет наличие формулы;

IsAnyText = IsHeader1 $\vee$ IsHeader2 $\vee$ IsHeader3 $\vee$ $\vee$ IsHeader4 $\vee$ IsHeader5 $\vee$ IsText определяет наличие заголовков (1—5 уровней) или обычного текста.

Указанная последовательность условий позволяет наиболее точным образом определять элементы, пошагово исключая доступные типы объектов. Алгоритм, который определяет наличие именованной иллюстрации в текущем абзаце документа, представлен на Листинге 2.

Листинг 2. Пример определения наличия в документе Microsoft Word именованной иллюстрации (функция IsPictureWithAnnotation, язык — VBA)

```
' В целях иллюстрации подхода, значения параметров стиля синтаксического анализа,
' приведенные выше в Листинге 1, указаны непосредственно в коде

Function IsPictureWithAnnotation(CurrentParagraph As Paragraph) As Boolean

     IsPictureWithAnnotation = False

     ' 1.1) Определение типа элемента на основании объектной модели Word
     If CurrentParagraph.Range.InlineShapes.Count = 1 Or _
     CurrentParagraph.Range.ShapeRange.Count = 1 Then

     ' 1.2) Определение абзаца с аннотацией к иллюстрации и
     '      его проверка на соответствие стилю синтаксического анализа
     Dim AnnPar As Paragraph
     Set AnnPar = GetAnnotationParagraph(CurrentParagraph, "Auto")

     IsPictureWithAnnotation = Not (AnnPar Is Nothing)

   End If
End Function
Public Function GetAnnotationParagraph(CurrentParagraph As Paragraph, _
   AnnPos As String) As Paragraph

   ' AnnPos — место расположения подписи к рисунку
   ' Above — над, Below — под, Auto — определяется автоматически

   ' Здесь также может осуществляться проверка других параметров аннотации

Set GetAnnotationParagraph = Nothing

   If AnnPos = "Above" Then
      If CheckPictureAnnotationMatch(CurrentParagraph.Previous) Then _
        Set GetAnnotationParagraph = CurrentParagraph.Previous

   ElseIf AnnPos = "Below" Then
      If CheckPictureAnnotationMatch(CurrentParagraph.Next) Then _
        Set GetAnnotationParagraph = CurrentParagraph.Next

   ElseIf AnnPos = "Auto" Then
      Dim AnnPar As Paragraph

      Set AnnPar = GetAnnotationParagraph(CurrentParagraph, "Above")
      If Not AnnPar Is Nothing Then
           Set GetAnnotationParagraph = AnnPar
      Else
       Set GetAnnotationParagraph =
           GetAnnotationParagraph(CurrentParagraph, "Below")
      End If
   End If

End Function

Function CheckPictureAnnotationMatch(CurrentParagraph As Paragraph) As Boolean
   ' Для проверки соответствия маске используются регулярные выражения

   Set re = New RegExp
   re.IgnoreCase = False
   re.Pattern = "^(Рисунок|Иллюстрация|Image|Рис\.)(.*)(.*)"
   CheckPictureAnnotationMatch = re.Test(CurrentParagraph.Range.Text)
End Function
```

Результатом работы модуля синтаксического анализа является создание NK-документа. Такой документ внутреннего формата включает информацию о документе в целом, а также о его составных элементах и их параметрах. Далее NK-документ передается в модуль проверки для дальнейшего анализа ВКР.

Модуль проверки. Вторым этапом общего алгоритма работы ИС является непосредственная проверка текста ВКР. На данном этапе происходит сравнение параметров переданного синтаксическим анализатором NK-документа с эталонными значениями стиля оформления. Стиль представляет собой набор правил, регламентирующих оформление документа и элементов его содержимого, включая оглавления, таблицы, заголовки и прочие объекты. Фрагмент примера стиля, определяющий внешний вид именованной иллюстрации в тексте документа, приведен в Листинге 3.

Листинг 3. Фрагмент примера стиля оформления, регламентирующий внешний вид именованной иллюстрации в тексте документа

```
{
    ...

    "PictureWithAnnotation": {
       "Picture": {
            "Font": {
                "Name": "Times New Roman",
                "Size": 14
            },
            "ParagraphFormat": {
                "Alignment": 1
            }
      },
      "Annotation": {
            "Font": {
                "Name": "Times New Roman",
                "Size": 12,
                "Bold": true
            },
            "ParagraphFormat": {
                "Alignment": 1,
                "LineSpacingRule": 1,
                "FirstLineIndent": 0,
                 "SpaceAfter": 17
             },
             "Position": "Below"
       }
    }
}
```

При наличии соответствующей опции на данном этапе также осуществляется внесение необходимых исправлений в исходный документ.

Модуль отчетности. На заключительном этапе работы ИС происходит генерация отчетов о результате проверки. В качестве шаблона вывода может использоваться стандартный образец бланка замечаний нормоконтролера [2] либо произвольный шаблон, обеспечивающий обязательное отражение следующей информации:

- ФИО студента, а также академическая информация (специальность, группа обучения и т. д.);
- содержание замечаний/исправлений по каждому из проверяемых разделов с указанием позиции ошибки (номера страницы или позиции специальной пометки) и требуемых/произведенных корректировок;
- дата проверки и ФИО нормоконтролера с указанием подписи.

Заключение

В статье рассмотрены особенности этапов проектирования и разработки ИС нормоконтроля ВКР. Предложенная архитектура программного обеспечения включает в себя модули синтаксического анализа, проверки и отчетности. В совокупности с концепцией стилей она позволяет разработать гибкий и современный программный продукт с масштабируемыми функциональными возможностями по анализу текста ВКР на предмет соблюдения принятых норм и правил их оформления. В рамках описания общей схемы работы системы предложен оригинальный алгоритм синтаксического разбора документа, позволяющий эффективно определять типы элементов для их дальнейшего комплексного анализа (проверки). Представлен вариант реализации этого алгоритма на основе объектной модели текстового процессора Microsoft Word.

Список литературы

1. **Петров Ю. И., Шупикова Ю. В.** Современные информационные системы нормоконтроля выпускных квалификационных работ // Программная инженерия. 2013. № 2. С. 36—41.
2. **Шупикова Ю. В., Петров Ю. И., Кунгурцева К. В., Гаранина Е. А.** Нормоконтроль выпускных квалификационных работ: сущность и необходимость автоматизации // Актуальные вопросы науки: Материалы VI Международной научно-практической конференции (10.07.2012). М.: Спутник+, 2012. С. 64—71.
3. **Delphi XE2.** Embarcadero Technologies. URL: http://edn.embarcadero.com/article/41593
4. **FireMonkey.** Embarcadero Technologies. URL: http://www.embarcadero.com /ru/products/firemonkey
5. **Introducing JSON.** URL: http://www.json.org/
6. **Спинеллис Д., Гусиос Г.** Идеальная архитектура. Ведущие специалисты о красоте программных архитектур. СПб.: Символ-Плюс, 2010. 528 с.
7. **Форд Н., Найгард М.** 97 этюдов для архитекторов программных систем. СПб.: Символ-Плюс, 2010. 224 с.
8. **Петров Ю. И., Шупикова Ю. В., Викулина Д. А., Макаров С. Н.** Способы и средства автоматизации текстового процессора Microsoft Word // Перспективы развития информационных технологий: сборник материалов VIII Международной научно-практической конференции / под общ. ред. С. С. Чернова. Новосибирск: Агентство "СИБПРИНТ", 2012. С. 98—107.
9. **Дьяченко А. В., Манжула В. Г., Попов А. Э., Семенихин И. Н., Толстобров А. П.** Построение информационных систем непрерывного образования на основе интернет-технологий. Пенза: Академия Естествознания, 2010. 130 с.
10. **Word** Object Model Overview. Microsoft Corporation. 2012. URL: http://msdn.microsoft.com/en-us/library/kw65a0we(v=vs.110).aspx
11. **Mansfield R.** Mastering VBA for Microsoft Office 2010. Indianapolis.: Wiley Publishing, Inc, 2010. 880 p.
12. **LibreOffice.** The Document Foundation. URL: http://www.libreoffice.org/
13. **ISO/IEC 26300:2006** — Information technology — Open Document Format for Office Applications (OpenDocument) v1.0. ISO. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485

УДК 002.53/55:001.814

А. В. Жарковский, канд. техн. наук, зам. проректора по научной работе,
e-mail: av.jarkov@yandex.ru,
А. А. Лямкин, канд. техн. наук, доц., e-mail: alex-ljamkin@yandex.ru,
С. А. Тревгода, стар. науч. сотр., e-mail: trevgoda@users.mns.ru,
Санкт-Петербургский государственный электротехнический университет "ЛЭТИ"

Структурный подход к автоматизации реферирования научно-технических текстов

Анализируются различные подходы к решению задачи автоматизации реферирования. Рассматривается формальная модель процесса построения структуры научно-технических текстов. Предложен критерий корректности текстовых структур и введены соответствующие структурные характеристики и ограничения.

Ключевые слова: *автоматизация, реферирование, функциональные отношения, структура текста, формализация*

На этапе создания и модернизации технических систем одной из наиболее важных задач является сбор, хранение, обработка и анализ большого объема научно-технической текстовой информации, представленной в электронном виде. В этой ситуации особенно актуальными становятся автоматизированные методы работы с большими объемами информации, в частности методы получения сжатого представления текстовых документов — рефератов и аннотаций.

Проведенный анализ работ в области автоматизации реферирования показал, что существует два основных подхода к решению данной задачи [1]:

- генерация реферата на основе использования методов искусственного интеллекта (абстракция);
- извлечение из исходного текста всех релевантных предложений (экстракция).

Реферирование путем абстракции использует сложные лингвистические алгоритмы на основе методов искусственного интеллекта. Выходом при этом является не просто набор предложений из исходного текста, а порождается новый текст (реферат), содержательно обобщающий первичные документы.

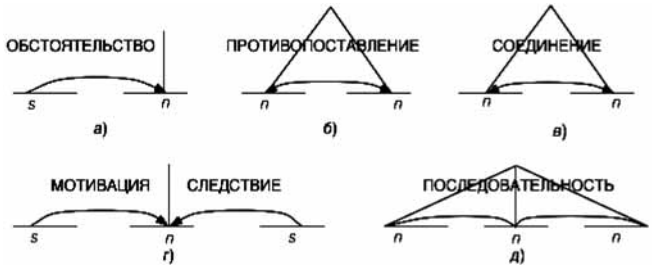
В этом случае для подготовки краткого изложения информации требуются высокопроизводительные вычислительные ресурсы, которые поддерживают системы обработки естественных языков, в том числе грамматики, и словари для синтаксического разбора и генерации естественно-языковых конструкций. Кроме того, для реализации этого метода нужны исчерпывающие онтологические справочники, отражающие соображения здравого смысла, и понятия, ориентированные на предметную область. Они необходимы для принятия решений во время анализа и определения наиболее важной информации. Перечисленные требования служат препятствием для широкого распространения подхода абстракции, как следствие, в настоящее время это направление представлено экспериментальными исследованиями и до широкой реализации еще не дошло.

При использовании подхода экстракции результат обработки одного или нескольких документов представляется как набор предложений. Среди этого набора система выбирает те, которые в наибольшей степени подходят под заданный критерий, т. е. являются более релевантными. Результатом является подмножество предложений исходного текста.

Исследования в области автоматического реферирования в настоящее время ведутся, главным образом, в рамках подхода экстракции с использованием статистических методов. Смысл этих методов заключается в отборе предложений с наибольшим весом, который рассчитывается на основе частоты появления слова в тексте или на основе месторасположения предложения в тексте, для включения их в реферат. Следует отметить, что при использовании вариаций статистических методов анализа не учитывается нелинейная и иерархическая природа текста, что неизбежно приводит к потере значимой информации при автоматическом формировании рефератов.

Текст не является линейной последовательностью единиц. Напротив, текст организован иерархично: элементарные единицы объединяются в единицы большего объема, те объединяются между собой и так до уровня целого текста. Для объединения единиц любого объема существует общий, единый набор структурных связей [2].

Структура текста определяется особенностями внутренней организации единиц текста и закономер-



Схемы функциональных отношений

ностями взаимосвязи этих единиц в рамках текста как цельного сообщения. Каждый текст имеет функционально-стилевую ориентацию (научный текст, художественный и др.) и обладает стилистическими качествами, которые диктуются данной ориентацией.

Научный стиль характеризуется логичностью и последовательностью изложения, обилием терминов, информационной насыщенностью заголовков параграфов и всего текста. Характерная для научного стиля специфика организации текста выражается в том, что каждый логически законченный отрывок текста обрамляется вступительными и заключительными предложениями, содержащими основную информацию в кратком виде. Учет особенностей научно-технических текстов позволяет провести формализацию их описания, выделить подмножество функциональных отношений и ключевых фраз, которые будут использоваться при разработке алгоритмов автоматического реферирования текста.

Теоретической основой предлагаемого метода формализованного описания научно-технического текста является теория риторической структуры текста (ТРС), которая рассматривает любой текст в виде древовидной структуры, узлы которой связаны между собой функциональными отношениями. Листьями дерева являются элементарные текстовые элементы (ЭТЭ), в данном случае являющиеся частью предложения.

Термин "функциональные" означает, что каждый фрагмент текста существует не сам по себе, а добавляется автором к другому с определенной целью. Функциональные отношения, как правило, являются асимметричными: более значимый их компонент называется *ядром* (*n*), менее значимый — *сателлитом* (*s*). Сателлит часто может быть опущен или заменен другим при сохранении функционального эффекта. В то же время, если опущено или изменено ядро, смысл текста и отношение существенно меняются.

Функциональные отношения могут выстраиваться в деревья на основе пяти структурных схем, которые представлены на рисунке. Большинство отношений соединяется по схеме *а*. Схема *г* покрывает случаи, в которых ядро соединено с несколькими сателлитами различными отношениями. Схемы *б*, *в* и *д* иллюстрируют мультиядровые отношения.

В соответствии с основными положениями ТРС:

- ЭТЭ представляют собой непересекающиеся части текста;
- функциональные отношения связывают текстовые элементы разного размера;
- ЭТЭ имеют в тексте различную значимость;
- структура текста может быть представлена в виде дерева.

При этом корректными структурами текста считаются такие, которые удовлетворяют следующим ограничениям:

- функциональные структуры являются деревьями, в которых элементы одного уровня представляют собой непрерывный текст;
- элементы могут быть двух типов — ядро и сателлит;
- каждый текстовый элемент может быть связан с другим только одним отношением.

С учетом этих ограничений структуры текста представляют собой деревья, смежные узлы которых — непрерывный текст.

Анализ основных положений ТРС показывает, что ее непосредственное применение для автоматизации процесса реферирования невозможно ввиду следующих недостатков:

- нет формальной спецификации, которая позволила бы отличить корректное дерево от некорректного;
- нет алгоритмов для построения таких деревьев.

В связи с этими обстоятельствами необходимо выполнить уточнение и дополнение этих положений для описания структуры научно-технического текста.

Предлагаемый метод формализованного описания структуры текста включает в себя определения:

— критерия корректности структуры текста;
— характеристик, описывающих структуры текста;
— ограничений на корректные структуры текста.

В ТРС нет формальных правил для объединения двух смежных фрагментов текста в один. Однако существование понятий ядра и сателлита и их роли в тексте позволяют сделать некоторые допущения для такого объединения. Как было отмечено ранее, то, что выражается во фрагменте-ядре в функциональном отношении более важно нежели то, что содержится во фрагменте-сателлите. Этот факт означает, что удаление всех фрагментов-ядер из текста приведет к тому, что смысл текста потеряется.

На основе анализа корпуса научно-технических текстов на русском языке выдвигается предположение, что функциональные отношения, которые лежат между большими фрагментами текста и представляют большие деревья, могут быть обусловлены существованием функциональных отношений между поддеревьями до их объединения в бóльшее дерево. Данное предположение вытекает из того факта, что при анализе различных текстов выяснилось, что для определения типов функциональных отношений между большими фрагментами текста, а также для определения границ этих фрагментов часто было удобным ассоциировать некие уникальные абстракции с данными фрагментами. Таким образом, вместо того чтобы оценивать, лежит ли функциональное отношение между двумя большими фрагментами текста, часто было проще ассоциировать некоторые уникальные абстракции с данными фрагментами и затем пытаться определить отношение между этими абстракциями. Эти уникальные абстракции в большинстве случаев относились к фрагментам-ядрам, принадлежащим бóльшим фрагментам текста.

Таким образом, на основе анализа корпуса научно-технических текстов на русском языке, а также работ [3, 4] представляется целесообразным ввести следующий критерий корректности структуры текста: *если функциональное отношение лежит между двумя текстовыми элементами структуры текста, тогда оно же лежит между, по крайней мере, двумя ключевыми ЭТЭ-потомками этих элементов.*

Необходимо отметить, что ключевыми ЭТЭ являются те, которые играют роль ядра в функциональном отношении. Основная идея этого критерия заключается в том, что ЭТЭ-ядра играют бóльшую роль в тексте, нежели ЭТЭ-сателлиты и, в принципе, при удалении всех сателлитов смысл текста должен сохраниться. Если применить этот принцип рекурсивно ко всему тексту, представляя его в виде дерева, получим дерево, удовлетворяющее критерию.

Следующим этапом формализации является определение характеристик структуры текста, которые связаны с каждым ЭТЭ и дают достаточную информацию для полного описания текстовой структуры и дальнейшей алгоритмизации процесса автоматического реферирования. В качестве таких характеристик выбираются следующие:

- $S(l, h, status)$ — показывает статус узла, который может иметь значения *NUCLEUS*(*ЯДРО*), *SATELLITE*(*САТЕЛЛИТ*) или *NONE*(*НЕОПРЕДЕЛЕН*), где l — индекс левого ЭТЭ; h — индекс правого ЭТЭ; *status* — значение статуса узла;
- $T(l, h, relation\_name)$ — показывает имя функционального отношения, которое лежит между своими прямыми потомками, где *relation_name* — функциональное отношение;
- $P(l, h, unit\_name)$ — показывает имя ключевого ЭТЭ на основе своих прямых потомков, где *unit_name* — название или индекс ЭТЭ.

Заключительным этапом формализации является определение ограничений для построения корректных структур текста. Текст представляет собой множество ЭТЭ, которые являются листьями дерева, а фрагменты текста являются узлами более верхнего уровня и состоят из нескольких ЭТЭ.

Пусть имеем текст из N ЭТЭ, где $[l, h]$ является его фрагментом, причем l и h — левый и правый индексы ЭТЭ соответственно**.**

Для генерации только корректных структур необходимо ввести следующие ограничения для текста из N ЭТЭ.

- Для каждого фрагмента $[l, h]$ предикат S имеет домен значений *NUCLEUS*, *SATELLITE*, *NONE*. Для случая, когда $l = h$, эти значения могут быть только *NUCLEUS*, *SATELLITE*:

$$[(1 \leqslant h \leqslant N) \wedge (1 \leqslant l \leqslant h)] \rightarrow$$
$$\rightarrow \{[l = h \rightarrow (S(l, h, NUCLEUS) \vee S(l, h, SATELLITE))] \wedge$$
$$\wedge [l \neq h \rightarrow (S(l, h, NUCLEUS) \vee S(l, h, SATELLITE) \vee$$
$$\vee S(l, h, NONE))]\}.$$

- Статус любого фрагмента уникален:

$$[(1 \leqslant h \leqslant N) \wedge (1 \leqslant l \leqslant h)] \rightarrow$$
$$\rightarrow [(S(l, h, status_1) \wedge S(l, h, status_2)) \rightarrow status_1 = status_2.$$

- Для каждого фрагмента $[l, h]$ предикат T имеет домен значений в виде множества функциональных отношений, соответствующих этому фрагменту:

$$[(1 \leqslant h \leqslant N) \wedge (1 \leqslant l \leqslant h)] \rightarrow$$
$$\rightarrow \{[l = h \rightarrow T(l, h, LEAF)] \wedge [l \neq h \rightarrow (T(l, h, NONE) \vee$$
$$\vee (T(l, h, name) \rightarrow relevant\_rel(l, h, name)))]\},$$

где *relevant_rel*$(l, h, name)$ означает множество отношений, лежащих между фрагментами текста внутри $[l, h]$; *LEAF* — лист структуры текста.

- По крайней мере одно функциональное отношение лежит между двумя смежными фрагментами:

$$[(1 \leqslant h \leqslant N) \wedge (1 \leqslant l \leqslant h)] \rightarrow$$
$$\rightarrow [(T(l, h, name_1) \wedge T(l, h, name_2)) \rightarrow name_1 = name_2].$$

- Для каждого фрагмента $[l, h]$ предикат P имеет домен значений в виде множества ЭТЭ, из которых он состоит:

$$[(1 \leqslant h \leqslant N) \wedge (1 \leqslant l \leqslant h)] \rightarrow$$
$$\rightarrow [P(l, h, NONE)] \vee P(l, h, u) \rightarrow relevant\_rel(l, h, u))],$$

где u — функциональное отношение.

- Текстовые фрагменты не пересекаются:

$$[(1 \leqslant h_1 \leqslant N) \wedge (1 \leqslant l_1 \leqslant h_1) \wedge (1 \leqslant h_2 \leqslant N) \wedge$$
$$\wedge (1 \leqslant l_2 \leqslant h_2)] \wedge (l_1 < l_2) \wedge (h_1 < h_2) \wedge (l_2 \leqslant h_1)] \rightarrow$$
$$\rightarrow [\neg S(l_1, h_1, NONE) \rightarrow S(l_2, h_2, NONE)].$$

- Текстовый фрагмент со статусом *NONE* не участвует в результирующем дереве:

$$[(1 \leqslant h \leqslant N) \wedge (1 \leqslant l \leqslant h)] \rightarrow$$
$$\rightarrow [(S(l, h, NONE) \wedge P(l, h, NONE) \wedge T(l, h, NONE)) \rightarrow$$
$$\rightarrow (\neg S(l, h, NONE) \wedge \neg P(l, h, NONE) \rightarrow$$
$$\rightarrow \neg T(l, h, NONE))]$$

- Существует главный фрагмент, корень дерева, который покрывает весь текст:

$$(\neg S(l, N, NONE) \wedge \neg P(l, N, NONE) \rightarrow$$
$$\rightarrow \neg T(l, N, NONE).$$

Таким образом, предложенный критерий корректности структуры текста и выполненная формализация характеристик и ограничений на корректные структуры являются расширением процесса формализации основных положений РТС. Они определяют условия объединения фрагментов текста, позволяют минимизировать набор необходимых параметров, достаточных для полного описания структуры текста, и способны существенно уменьшить избыточность порождаемых альтернативных структур текста. Представленная формальная модель позволяет перейти к разработке алгоритмов автоматизации реферирования.

Список литературы

1. **Сабинин О. Ю., Тревгода С. А.** Системы автоматического реферирования текста // Приборы и системы. Управление, контроль, диагностика. 2008. № 1. С. 23—26.

2. **Заболеева-Зотова А. В., Камаев В. А.** Лингвистическое обеспечение автоматизированных систем. М.: Высшая школа, 2008. 245 с.

3. **Mann W. C., Thompson S. A.** Rhetorical structure theory: Toward a functional theory of text organization // Text — Interdisplinary Journal for the Study of Discourse. 2009. N 8. P. 243—281.

4. **Interjeet M.** Advances in automatic text summarization. Cambridge: The MIT Press, 1999. 442 p.

УДК 66.011

Ф. П. Соколов, канд. техн. наук, ген. директор,
И. Н. Сизых, зав. комм. отд.,
А. В. Сухова, гл. инж. проекта,
Институт по проектированию производств органического синтеза "Гипросинтез",
г. Волгоград,
e-mail: info@giprosintez.ru

Компьютерные моделирующие комплексы для проектирования производственных объектов газо- и нефтехимической переработки

Рассмотрены различные программные комплексы, которые можно использовать на этапе проектирования производственных объектов газонефтехимического профиля. Отмечено, что программный комплекс Aspen ONE является одним из немногих программных продуктов, содержащих набор приложений для технического и экономического анализа существующих и комплексного моделирования новых технологий и производств, он также имеет отдельный блок для моделирования оборудования, которое не может быть описано стандартными математическими моделями, представленными в библиотеках программного комплекса. Описаны особенности применения программного комплекса Aspen ONE на этапе проектных исследований по реконструкции существующего производства трихлорсилана в г. Усолье-Сибирское Иркутской области.

Ключевые слова: *программные комплексы, проектирование промышленных производств газонефтехимического профиля, программный продукт Aspen ONE, модуль Aspen Custom Modeler, реконструкция производства трихлорсилана*

Введение

Применение современных компьютерных моделирующих комплексов на любом этапе жизненного цикла производственного объекта позволяет без существенных материальных и временных затрат проводить проектные исследования процессов, учитывать влияние внешних факторов (например, изменение состава сырья, изменение требований к конечным и промежуточным продуктам) на показатели действующих производств. Особенно большое значение компьютерное моделирование имеет на этапе проектирования для повышения качества проектных исследований и сокращения сроков проектирования производственных объектов [1].

Под проектным исследованием понимается выполнение комплекса инженерных расчетных и графических работ с использованием математического моделирования в целях получения информации об изучаемом методе производства применительно к

условиям реализации его в промышленном масштабе [2]. В настоящее время выполнение проектных исследований невозможно без использования современных компьютерных комплексов.

К наиболее важным преимуществам компьютерного моделирования технологических процессов относятся: расчетные исследования и анализ результатов для выбора оптимального варианта технологического процесса; определение оптимальных режимов работы оборудования для получения желаемой производительности оборудования; учет влияния характеристик сырья, сбоев в работе и остановки оборудования на безопасность и надежность процесса производства.

Основные программные комплексы для разработки и проектирования новых производственных объектов

В основу средств моделирования технологических процессов заложены общие принципы расчетов материально-тепловых балансов химических производств (связанных с изменением агрегатного состояния, компонентного и химического составов материальных потоков) [3].

Анализ существующих программных комплексов показывает, что система моделирования включает набор следующих основных подсистем, обеспечивающих решение задачи моделирования химико-технологических процессов: базы данных термодинамических свойств чистых компонентов; средства представления свойств исходных веществ; методы расчета термодинамических свойств; набор моделей для расчета отдельных элементов технологических схем — процессов; средства для формирования технологических схем из отдельных элементов; средства для расчета технологических схем.

Моделирующие системы позволяют проводить расчеты в реальном времени, оперативно предоставляя их результаты. Это дает возможность лучше понять сущность моделируемых процессов. Можно собрать и испытать разные схемы, исследовать пусковые режимы, получить представление о реально работающем процессе и поведении объекта в нештатных ситуациях, о влиянии изменения рабочих параметров на качество продуктов.

В настоящее время общепризнанными лидерами мирового IT-рынка в рассматриваемом производственном секторе считаются продукты двух компаний: Invensys Inc. (Simulation Sciences Esscor входит в состав Invensys) и Aspen Technologies [3]. Канадская компания Hyprotech Ltd. (в 2004 г. вошла в состав Aspen Technologies) создала программные комплексы HYSYS и Hysim, которые позволяют выполнять статическое моделирование практически всех основных процессов газопереработки, нефтепереработки и нефтехимии. Особый акцент в них сделан на работу с уравнением состояния Пенга-Робинсона. Реализующая его программа имеет расширенный набор модификаций этого уравнения, включающих работу с несимметричными коэффициентами бинарного взаимодействия и различными правилами смещения, модификации для работы с водой, гликолями и аминами. Она реализует оригинальный алгоритм расчета ректификационных колонн и практически не имеет ограничений в отношении набора задаваемых спецификаций и сложности колонны. Программный продукт осуществляет табличный ввод данных, по которому затем строится графическое изображение схемы, с возможностью экспорта в AutoCAD. Дополнительный пакет Hyprop позволяет эффективно обрабатывать экспериментальные данные по свойствам чистых компонентов и затем использовать полученные корреляции в расчетах.

Наряду с возможностью статического моделирования технологических схем, пакет HYSYS позволяет в той же среде проводить динамическое моделирование отдельных процессов и всей технологической цепочки, а также разрабатывать и отлаживать схемы регулирования процессов. Представляется возможность выполнять расчеты основных конструктивных характеристик сепарационного оборудования, емкостей, теплообменной аппаратуры, тарельчатых и насадочных ректификационных колонн. Программа имеет развитый графический интерфейс и хорошо интегрирована с офисными приложениями Microsoft.

Программные продукты Pro II и ProVision разработаны американской фирмой Simulation Sciences Inc. В Pro II/ProVision заложены возможности моделирования широкого спектра химических и нефтехимических производств. Существуют возможности для работы с растворами электролитов, проведения гидравлических расчетов сепарационного оборудования, реакторов, насадочных и тарельчатых ректификационных колонн. Разработан комплекс для динамического моделирования — Protiss. Фирма предлагает пакет моделирования гидравлики нефтегазовых месторождений, систем сбора и транспорта нефти и газа — Pipeface [4].

Кроме перечисленных выше программных продуктов, разработаны программные средства для инженерного моделирования, которые предоставляют пользователю значительно меньшие, но достаточные возможности для решения основных задач инженера-проектировщика и инженера-технолога. Среди таких программных продуктов следует выделить CHEMCAD III, PROSIM, DESIGN II.

Программный пакет CHEMCAD III разработан фирмой ChemStations Inc. и включает средства статического моделирования основных процессов, осно-

ванных на фазовых и химических превращениях. С его помощью возможен расчет геометрических размеров и конструктивных характеристик основных аппаратов, а также оценка стоимости оборудования [5].

Программный продукт PROSIM создан компанией Bryan Research & Engineering Inc. Он включает средства статического моделирования основных процессов газопереработки (включая гликолевую осушку, аминовую очистку, фракционирование) и нефтепереработки (атмосферно-вакуумная перегонка). Существуют средства для расчета геометрических размеров и конструктивных характеристик аппаратов [6].

Пакет DESIGN II компании WinSim Inc. имеет инструментарий для полноценного моделирования процессов в газонефтепереработке. Данный программный продукт включает базу данных по 880 компонентам, имеет интерфейсы Visual Basic, тесно интегрирован с Microsoft Excel [7].

Основными программными средствами, разработанными в Российской Федерации, являются КОМФОРТ и GIBBS [8]. Система моделирования КОМФОРТ представляет собой инструментальное средство для выполнения поверочных и проектных расчетов материально-тепловых балансов различных химических производств. Пакет КОМФОРТ состоит из управляющей программы и модулей расчета аппаратов. Управляющая программа с конкретным набором технологических модулей образуют предметно-ориентированную моделирующую программу, позволяющую выполнять расчеты для конкретного класса химико-технологических схем. Программа имеет средства для расчета всех основных процессов фракционирования для газопереработки. Интерфейс представляет собой систему с табличным кодированием данных [9].

Программный пакет GIBBS включает средства для моделирования процессов промысловой подготовки природных газов, установок низкотемпературной сепарации и низкотемпературных детандерных заводов с частичным или полным фракционированием жидких углеводородов, процессов промысловой и заводской подготовки и переработки газоконденсата и нефти, деэтанизации, стабилизации и фракционирования по топливному варианту, газофракционирования.

Программный комплекс имеет средства расчета:

- синтеза нефтяной смеси по данным лабораторных анализов;
- товарных свойств фракций моторных топлив, условий образования и ингибирования газовых гидратов;
- дифференциальной конденсации пластовых смесей;
- условий образования твердой фазы CO_2.

В пакете есть и другие утилиты инженерного применения [3].

Программный комплекс Aspen ONE

Американская компания Aspen Technologies Inc. разработала программный комплекс Aspen ONE, который позволяет проводить предпроектные исследования и выполнять технологические расчеты при проектировании производственных объектов. Основой данного программного комплекса является пакет моделирующих программ Aspen Plus. Данный программный комплекс, после приобретения Aspen Technology компании Hyprotech и объединения программных продуктов Aspen Plus и HYSYS в один пакет программ, является лидером рынка программного обеспечения для моделирования химико-технологических процессов. В России эти программные продукты начинают получать все большую известность [9].

Пакет Aspen Plus имеет развитый графический интерфейс и предназначен для моделирования процессов, основанных на химическом и фазовом превращении. Он реализует широкий набор необходимых для этого алгоритмов, который постоянно расширяется. Визуальный интерфейс Aspen Plus позволяет формировать технологические схемы непосредственно на экране компьютера, выбирая элементы из списка и соединяя их в определенном порядке (рис. 1, см. третью сторону обложки).

В результате создается технологическая схема процесса. Фрагмент технологической схемы процесса производства трихлорсилана, построенной средствами Aspen Plus, представлен на рис. 2 (см. третью сторону обложки).

Данный программный продукт позволяет пользователю не только моделировать химико-технологические процессы с использованием определенного набора моделей аппаратов (оборудования) из базы данных программного комплекса, но и с помощью модуля Aspen Custom Modeler моделировать отдельные технологические аппараты (единицы оборудования), которых нет в стандартных библиотеках программного комплекса. Программный комплекс позволяет также выполнять расчеты основных конструктивных характеристик и оценку стоимости оборудования.

Комплекс Aspen Dynamics представляет собой программную систему динамического моделирования технологических процессов и совместим на уровне данных с Aspen Plus. Сегодня эти программы объединены в новейшем интегрированном пакете Aspen ONE [10].

Программный комплекс Aspen ONE включает модули расчета статических процессов, которые позволяют прогнозировать поведение технологического процесса с учетом инженерных решений. Технологический процесс представлен в виде химических компо-

нентов, которые смешиваются, разделяются, нагреваются, охлаждаются и превращаются в продукты реакции на отдельных стадиях процесса. Данные передаются от модуля к модулю отдельно для каждой стадии процесса. Кроме этого, существуют инструментальные средства для детального моделирования процессов теплообмена. Отдельные модули позволяют моделировать процессы нефтегазопереработки, выполнять оптимизацию технологических параметров создаваемых процессов, а также получить экономическую оценку технологических установок, которые планируется использовать на проектируемых объектах.

Для оценки технико-экономических показателей создаваемых производств предназначен отдельный программный комплекс Aspen Process Economic Analyzer, позволяющий достаточно точно оценивать различные варианты инвестиций уже на ранних этапах разработки проекта и использующий в качестве исходных данных данные, полученные с помощью HYSYS, Aspen Plus, Pro II и CHEMCAD III.

С использованием этого пакета может быть обеспечен контроль поставок на строительную площадку необходимого оборудования и материалов (Aspen ONE Planning & Sheduling), а также тренинг для персонала (Aspen ONE Advanced Process Control). Всего программа включает более 60 отдельных модулей, позволяющих охватить практически весь процесс проектирования производственных объектов.

Важной особенностью пакета Aspen ONE является наличие модуля Aspen Energy Analyzer, позволяющего рассчитывать показатели энергоэффективности для проектируемых технологических процессов и их соответствия требованиям, регламентирующим выбросы CO_2 и других парниковых газов, а также модуля Aspen Optimizer для оптимизации уже действующих производств. Относительно новый модуль Aspen Properties Mobile в составе пакета обеспечивает быстрый доступ к информации о физических свойствах компонентов, перенося широкие возможности модуля Aspen Properties на мобильные устройства Apple iPad и iPhone.

В комплекте программ Aspen ONE содержится наиболее обширная база данных физико-химических свойств веществ, включая большую номенклатуру разных химических веществ, а также твердых материалов, электролитов, полимеров, кремнийорганических веществ. Наличие отдельных банков физико-химических свойств электролитов и полимеров является характерным преимуществом Aspen Plus перед программами-конкурентами.

Основной модуль Aspen Plus содержит большую библиотеку математических моделей технологических блоков, включающих разное число единиц оборудования, к которым относятся сепараторы и смесители, реакторное и теплообменное, насосное, компрессорное и колонное оборудование.

Пакет программ Aspen ONE имеет интерфейс обмена данными с программным комплексом трехмерного проектирования AVEVA PDMS (Plant Design Management System), который реализует одну из передовых технологий для трехмерного проектирования с централизованной системой хранения данных. Инструментальные средства пакета позволяют работать над сложными проектами и осуществлять автоматический выпуск чертежей и отчетов непосредственно из базы данных PDMS [10]. Такой обмен осуществляется с помощью модуля Aspen Basic Engineering. Интегрированный рабочий процесс с использованием различных программ, обменивающихся данными по единому стандарту, позволяет сократить трудозатраты и снизить число ошибок проектировщика при ручном вводе на 20 %, повышая таким образом качество проектирования [11].

Упоминавшийся ранее программный модуль Aspen Custom Modeler принимает на вход расчетные зависимости для моделируемого оборудования, написанные на различных языках программирования, в частности Intel Fortran 9.0 или 9.1, Microsoft Visual C++, Microsoft Visual Basic 6 (Aspen Custom Modeler Language). При работе Aspen Custom Modeler автоматически появляется текстовое окно, в которое заносится последовательность расчета на используемом языке программирования (изображение рабочего пространства программы Aspen Custom Modeler представлено далее в работе). Таким образом, модуль Aspen Custom Modeler позволяет на основе результатов физико-химических, кинетических и технологических исследований проектируемых процессов разрабатывать пользовательские математические модели технологических блоков, отсутствующие в стандартной базе данных программного комплекса.

В настоящее время в мире пользователями пакета программ Aspen ONE являются большие компании — лидеры в области нефтепереработки (BP, ExxonMobil, Shell, ConocoPhillips, Petrobras), химической (DuPont, Samsung, BASF, DOW, Mitsubishi Chemical, Bayer) и фармацевтической промышленности (Pfizer, Johnson & Johnson, GlaxoSmithKline). Следует, однако, отметить, что в России программный пакет Aspen ONE практически не освоен.

Анализ программных комплексов для проектирования новых производственных объектов и практический опыт проектирования химических производств показали, что моделирующий комплекс Aspen ONE, содержащий набор приложений для комплексного анализа и моделирования новых технологий и производств, является одним из немногих, позволяющих как моделировать и проектировать новые надеж-

ные и экономичные промышленные производства, так и подвергать всестороннему анализу действующие предприятия для разработки проектов их реконструкции. Именно поэтому для проектирования и выполнения полного комплекса технологических расчетов при проектировании производства трихлорсилана был использован программный комплекс Aspen ONE.

Применение программного комплекса Aspen ONE при проектировании производства трихлорсилана

Технология получения поликристаллического кремния из трихлорсилана является основной в производстве кремния для солнечных батарей. Такие батареи являются надежными экологически чистыми энергетическими системами. Развитие мировой солнечной фотоэнергетики связано с масштабными программами поддержки возобновляемой энергетики, которые реализуются в ряде стран Европы, а также в США и Японии.

Рост потребности в солнечной энергетике в последнее десятилетие привел к значительному расширению производства поликристаллического кремния во многих странах [12]. В России производственный комплекс поликристаллического кремния создается компанией НИТОЛ в г. Усолье-Сибирское Иркутской области при участии РОСНАНО, Сбербанка и Евразийского банка развития. В декабре 2010 г. произведена первая партия поликристаллического кремния [13]. Планируется построить еще семь заводов по производству поликристаллического кремния [14].

На этапе проектных исследований по реконструкции существующего производства трихлорсилана в г. Усолье-Сибирское Иркутской области был использован программный комплекс Aspen ONE. Далее представлены результаты отдельных проектных исследований, возникающие при этом трудности и способы их преодоления с помощью средств Aspen ONE.

Модель технологического процесса производства трихлорсилана состоит из технологических блоков, которые отвечают основным стадиям процесса: синтез трихлорсилана; сухая пылеочистка; мокрая пылеочистка; конденсация низкого давления; низкотемпературная конденсация, компримирование и разделение парогазовой смеси; стабилизация трихлорсилана-конденсата.

Для каждого технологического блока из стандартной библиотеки моделей программного комплекса были выбраны модели аппаратов, наиболее полно описывающие протекающие в блоке процессы, определены условия протекания процесса для каждой модели аппарата, заданы входящие в них материальные и тепловые потоки.

Составы и параметры всех внешних потоков технологической схемы (исходные реагенты) задаются в табличной форме, составы и параметры рассчитываемых потоков (продукты) представляются после расчета технологической схемы в той же табличной форме (рис. 3, см. четвертую сторону обложки).

Построенная технологическая схема с помощью Aspen ONE в целом достаточно подробно моделирует процесс производства трихлорсилана. Однако при использовании программного комплекса возникали определенные трудности при описании стадии очистки абгазов и моделировании процесса гидрохлорирования кремния в реакторе.

Для описания стадии очистки абгазов таблично задавали полный перечень веществ, участвующих в этом процессе, и использовали модель RadFrac — адсорбер, с указанием материальных потоков, характеризующих процесс на входе и выходе из аппарата очистки [15, 16]. При задании характеристик таких потоков на входе для каждого потока указывают температуру, давление, расход и состав.

В соответствии с исходными данными процесса материальные потоки в адсорбер описывались на вкладке Streams → AbgIn и Streams → Absorbent. Описание самого аппарата выполнялось на вкладке Blocks → ABS. Задавали 8 теоретических ступеней разделения (так как колонна содержит 13 тарелок, КПД тарелки принимали равным 0,60—0,65). На вкладке Setup → Streams описывались условия питания: поток абгазов подается снизу (на восьмую тарелку), поток воды — сверху (на первую тарелку). Соответственно, сверху отводятся очищенные абгазы, а снизу — отработанный сорбент. На вкладке Setup → Pressure задавали атмосферное давление по высоте колонны (0,001 МПа). Для используемой в процессе проектирования модели абсорбера на вкладке Blocks → ABS → Convergence обозначен алгоритм расчета и модель аппарата в виде Blocks → ABS → Reactions.

Химические реакции, протекающие в аппарате, описываются на вкладке Reactions → Reactions → R-1. Сложность при дальнейшем описании состояла в том, что необходимо выбрать тип реакции, а именно Equilibrium, Conversion или Kinetic. При задании типа реакции Equilibrium, несмотря на то что в ходе реакции образуются газообразное вещество (водород) и осадок (гелеобразная кремниевая кислота), в программе данная реакция рассматривается как обратимая. Более того, обратная реакция оказывается термодинамически выгодной.

При задании типа реакции Conversion возникает необходимость в описании уравнения конверсии по одному из реагентов. Однако такая информация отсутствует в исходных данных на процесс и ее крайне мало в литературных источниках. При задании типа

реакции Kinetic необходимо описывать систему кинетических уравнений. Информация по кинетике реакции на стадии очистки абгазов также отсутствует. После анализа всех имеющихся данных был выбран тип реакции Conversion. Необходимая для этого информация была получена на основе анализа соответствующей литературы и по соображениям инженерно-эвристического характера с учетом выхода целевого продукта. После того как вся исходная информация была определена и введена в систему, выполнялся расчет. Результаты расчета отображаются на вкладке Blocks → ABS → Stream Result.

Проведенные исследования показали, что с использованием стандартных моделей аппаратов (реакторов), имеющихся в пакете Aspen ONE, нельзя описать процесс синтеза трихлорсилана в реакторе с псевдоожиженным слоем. Такие модели реакторов колонного типа RStoic (реактор идеального смещения непрерывного действия, реактор идеального смещения периодического действия, реактор идеального вытеснения) не учитывают особенностей протекания процесса в условиях псевдоожижения [17, 18]. Использование в качестве модели реактора той, которая предлагается программным комплексом, а именно модели процесса сушки в псевдоожиженном слое, также к должному результату не привело. Анализ особенностей протекания процесса в этих условиях показал, что данная модель не может быть использована для описания реактора с псевдоожиженным слоем, так как она не учитывает химических реакций, протекающих в процессе синтеза трихлорсилана.

Таким образом, для моделирования процесса синтеза трихлорсилана в псевдоожиженном слое появилась необходимость разработать новую модель реактора. На основе такой модели был составлен алгоритм расчета реактора кипящего слоя. Схема алгоритма представлена на рис. 4. С помощью программного модуля Aspen Custom Modeler была создана программа для моделирования реактора с псевдоожиженным слоем.

При работе с Aspen Custom Modeler автоматически появляется текстовое окно, в котором представлен расчет на языке программирования Aspen Custom Modeler Language. Изображение рабочего пространства программы Aspen Custom Modeler представлено на рис. 5 (см. четвертую сторону обложки).

Таблица спецификации, куда вносят исходные данные и где отображаются результаты расчета, представлена на рис. 5 (см. четвертую сторону обложки). Здесь в окне ввода данных и вывода результатов: *free* — расчетный параметр (результат расчета); *fixed* — фиксированный параметр (исходные данные, константы). На рис. 6 представлены отдельные фрагменты выполнения расчетов по программе для моделирования реактора с псевдоожиженным слоем.

В результате проведенных расчетов получены значения начальной высоты псевдоожиженного слоя (H_0 =

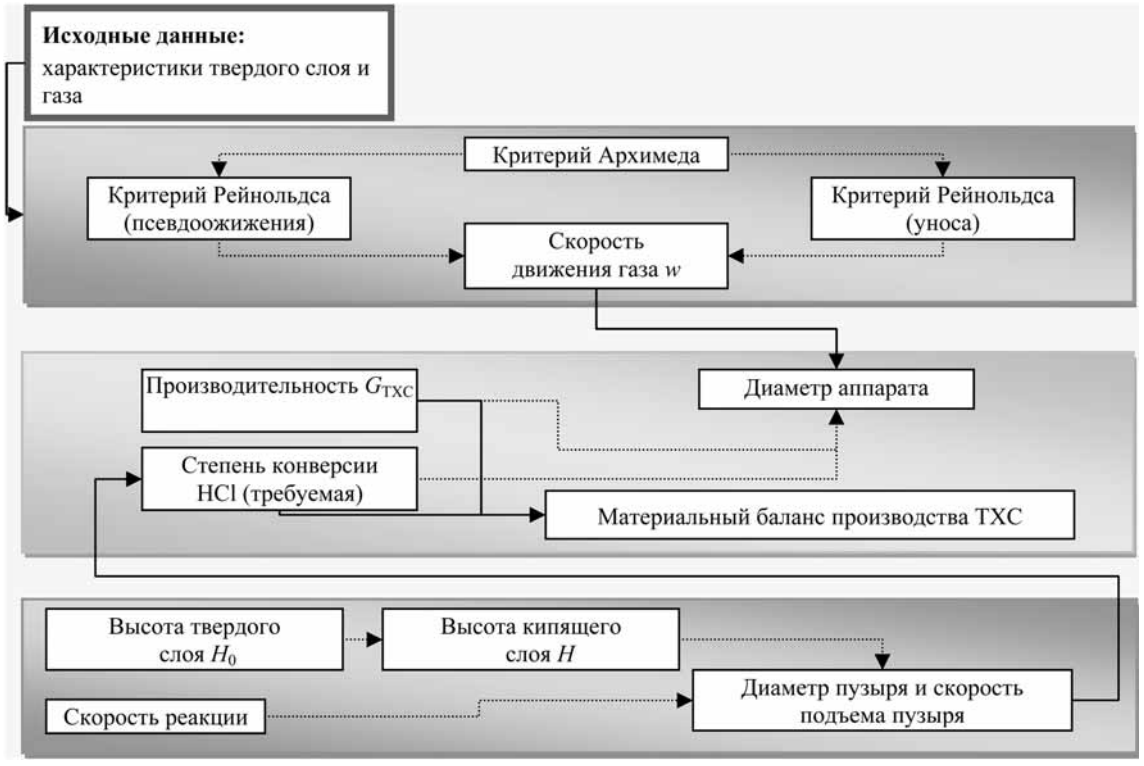


Рис. 4. Схема алгоритма расчета реактора кипящего слоя: ТХС — трихлорсилан

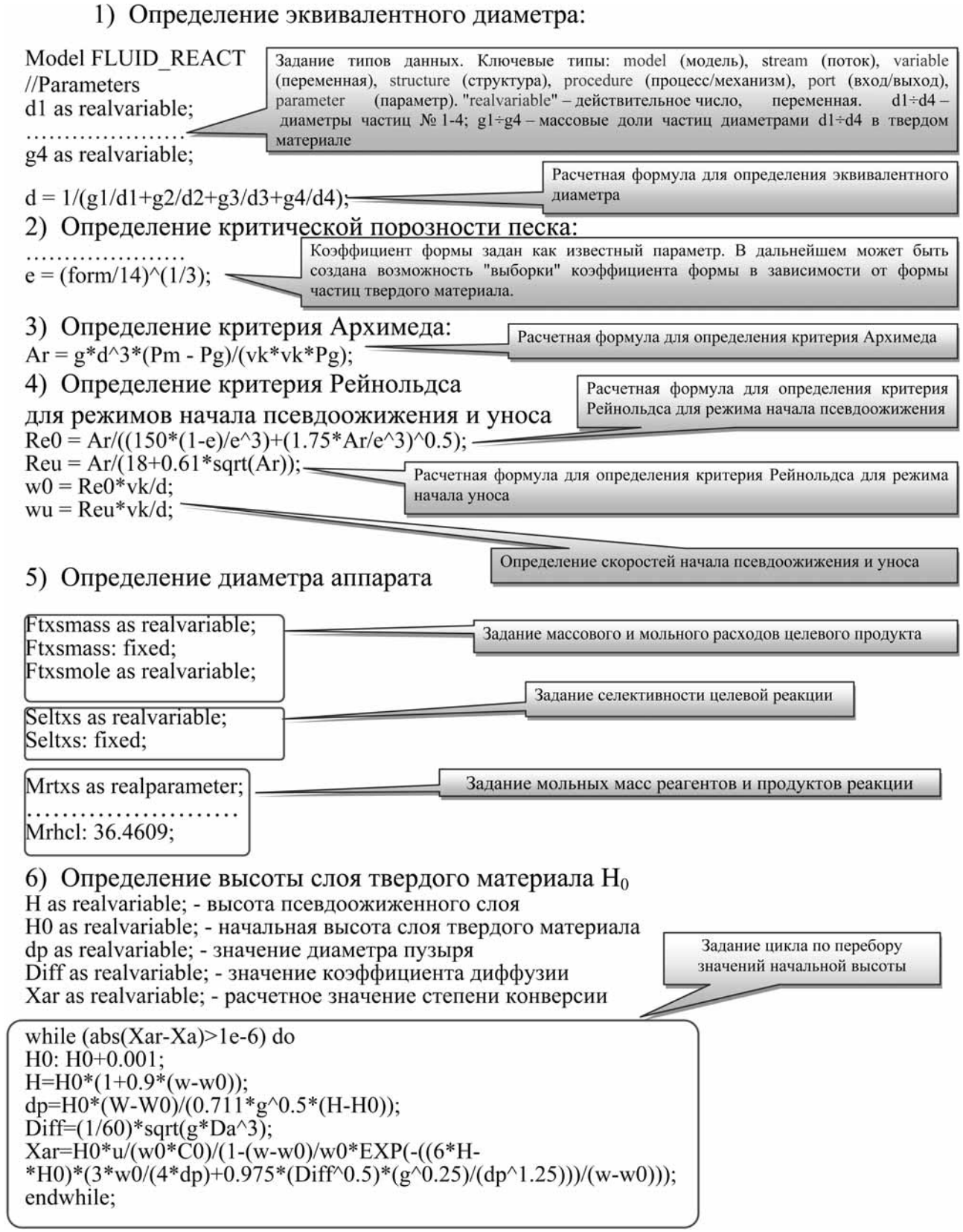


Рис. 6. Фрагменты выполнения расчетов по программе для моделирования реактора с псевдоожиженным слоем

= 10,4 м) и достигнута степень конверсии (X = 95 % по кремнию), соответствующая заданной. Разработанная математическая модель для моделирования реактора с псевдоожиженным слоем была реализована программно и добавлена в библиотеку комплекса Aspen Plus.

Заключение

В настоящей работе рассмотрены программные комплексы, которые используются на этапе проектирования современных промышленных производств газонефтехимического профиля, включающих сложные технологические системы. Отмечено, что Aspen ONE является уникальным по своим возможностям программным продуктом, который предоставляет средства для моделирования химико-технологических процессов в рассматриваемой предметной области. С помощью модуля Aspen Custom Modeler этот программный комплекс позволяет моделировать отдельные технологические аппараты (единицы оборудования), которые изначально не предусмотрены в его стандартных библиотеках. Комплекс включает более 60 отдельных модулей, позволяющих охватить практически весь процесс технологических расчетов, необходимых при проектировании производственных объектов. Пакет программ Aspen ONE имеет интерфейс обмена данными с программной системой трехмерного проектирования PDMS.

Описаны особенности применения программного комплекса Aspen ONE на этапе проектных исследований по реконструкции существующего производства трихлорсилана в г. Усолье-Сибирское Иркутской области. Определено, что стандартные модели реакторов колонного типа RStoic и модель процесса сушки в псевдоожиженном слое, представленные в библиотеке программного комплекса, не учитывают особенностей протекания процесса в условиях псевдоожижения. На основе приложения Aspen ONE (модуль Aspen Custom Modeler) для моделирования процесса синтеза трихлорсилана в реакторе с псевдоожиженным слоем разработаны соответствующая математическая модель и алгоритм, программная реализация которых добавлена в библиотеку программ комплекса Aspen Plus.

Практический опыт проектирования производства трихлорсилана показал, что программный комплекс Aspen Plus обеспечивает согласованность данных для моделирования химических и нефтехимических процессов и их инженерных расчетов. Он способствует сокращению времени на реализацию проекта, делает процесс проектирования более удобным и может быть рекомендован к использованию на практике.

Список литературы

1. **Соколов Ф. П.** Внедрение и адаптация технологий трехмерного проектирования — опыт ООО "Гипросинтез" // Труды Международной конференции "Системы проектирования, технологической подготовки производства и управления этапами жизненного цикла промышленного продукта". Москва, 2008. С. 7—10.
2. **Гуревич Д. А.** Проектные исследования химических производств. М.: Химия, 1976. 208 с.
3. **http://www.technoil.ru/reviews/Amodeling-review.htm#A_00.**
4. **http://www.simsci-esscor.com.**
5. **http://www.chemstations.com.**
6. **http://www.bre.com.**
7. **http://www.winsim.com.**
8. **Волин Ю., Островский Г.** Анализ гибкости — новый этап в компьютерном моделировании химико-технологических систем // The Chemical Journal. 2002. P. 50—53.
9. **http://www.oil-industry.ru.**
10. **http://www.aspentech.ru.**
11. **http://www.aveva.com**
12. **Яркин В. Н., Кисарин О. А., Реков Ю. В., Червонный И. Ф.** Кремний для солнечной энергетики: конкуренция технологий, влияние рынка, проблемы развития // Теория и практика металлургии. 2010. № 1. С. 114—125.
13. **http://www.nitol.ru.**
14. **http://www.russianelectronics.ru.**
15. **Холоднов В. А., Викторов В. К., Ананченко И. В., Чепикова В. И.** Лабораторный практикум по моделированию и оптимизации химико-технологических процессов с помощью программного продукта Aspen PLUS: учебное пособие. СПб.: Изд-во СПбГТИ, 2000. 41 с.
16. **Дворецкий С. И., Матвеев С. В., Путин С. Б., Туголуков Е. Н.** Основы математического моделирования и оптимизации процессов и систем очистки и регенерации воздуха: учебное пособие. Тамбов: Изд-во Тамбовского государственного технического ун-та, 2008. 324 с.
17. **Касаткин А. Г.** Основные процессы и аппараты химической технологии. М.: Химия, 1971. 784 с.
18. **Левеншпиль О.** Инженерное оформление химических процессов. М.: Химия, 1969. 648 с.

ИНФОРМАЦИЯ

УДК 004.75+004.41

А. С. Шундеев, канд. физ.-мат. наук, вед. науч. сотр.,
НИИ механики МГУ имени М. В. Ломоносова,
e-mail: alex.shundeev@gmail.com

Введение в стандарт IEEE 754

Представлено введение в машинную арифметику чисел с плавающей точкой, а также в соответствующий стандарт IEEE 754. В основу статьи положены материалы, которые использовались автором при проведении практических занятий по курсу "Работа на ЭВМ и программирование" для первоначального ознакомления учащихся с данной предметной областью.

Ключевые слова: *арифметика чисел с плавающей точкой, IEEE 754*

Введение

Машинная арифметика чисел с плавающей точкой [1] играет важную роль в учебных курсах по программированию и может рассматриваться как фундамент для такой дисциплины, как численные методы. Вместе с тем необходимо отметить, что многие подходы и методы, возникающие в этой области, являются трудными и неочевидными для рядового учащегося.

Например, уже в самых простейших программах приходится оперировать с вещественными числами, и возникает необходимость в их сравнении на равенство. Если в программе на языке C [2, 3] воспользоваться операциями == или != для вещественных чисел (`float` или `double`), то популярный компилятор `gcc`, скорее всего, выдаст предупреждающее сообщение вида

warning: comparing floating point with == or != is unsafe,

говорящее о небезопасности применения подобных операций. Это связано с тем, что для большинства вещественных чисел отсутствует их точное машинное представление. В силу конечности памяти компьютера вместо числа x программа оперирует его некоторым приближением вида $x + \delta$, где δ — погрешность приближения. Кроме того, результат выполнения арифметических операций также несет в себе некоторую погрешность, которая может накапливаться и расти. Подобное объяснение можно дать учащемуся, не вдаваясь в подробности машинного представления вещественных чисел.

Приведенное объяснение может быть закреплено путем написания и запуска программы, оперирующей с вещественными числами, в которой рост погрешности может быть явно отслежен. В качестве подобной программы можно использовать следующую. Реализуются две процедуры вычисления суммы N первых членов арифметической прогрессии. Первая процедура вычисляет сумму по определению (путем суммирования), а вторая — по известной формуле. Программа на вход получает арифметическую прогрессию (пару — начальный член и разница). Программа должна определить для какого N модуль разницы сумм, вычисленных первым и вторым способами, будет отличаться на заданное значение, например 10^{-7}.

Другой интересный пример разобран в книге [1]. Рассматривается последовательность чисел u_0, u_1, ..., u_n, ..., где

$$u_n = 111 - 1130/u_{n-1} + 3000/u_{n-1}u_{n-2}.$$

При $u_0 = 2$ и $u_1 = -4$ предел данной последовательности равен 6. При других начальных значениях предел может быть равен 100. Несмотря на точное машинное представление чисел 2 и -4, в результате возникновения погрешности при выполнении арифметических операций программа, вычисляющая предел данной последовательности, в качестве результата выдаст число, близкое к 100.

Приведенные примеры и рассуждения носят вводный характер. Более подробное изучение машинной арифметики чисел с плавающей точкой связано с рассмотрением стандарта IEEE 754 [4], которое будет проведено в основной части статьи.

Данная статья носит методический характер и создана на основе материалов, которые автор использовал при проведении практических занятий по курсу *"Работа на ЭВМ и программирование"*.

Формат чисел двойной точности

Стандарт IEEE 754 [4] является основой современной машинной арифметики чисел с плавающей точкой и имеет длительную историю развития [1]. Стан-

дарт описывает форматы представления машинных чисел (мантиссы, показателя экспоненты и знака), а также требования к преобразованиям чисел в процессе выполнения математических операций.

В качестве основного объекта будем рассматривать вещественные числа двойной точности (`double`), имеющие размер 64 бита. Битовое представление этих чисел имеет следующий формат:

$$s\ p_{10}\ p_9\ \ldots\ p_1\ p_0\ m_{51}\ m_{50}\ \ldots\ m_1\ m_0.$$

Старший (63-й бит) s задает знак числа. Биты $p_{10} \ldots p_0$ (с 62-го по 53-й бит включительно) определяют показатель экспоненты, а биты $m_{51} \ldots m_0$ (с 52-го по 0-й бит) определяют мантиссу. Подобное битовое представление однозначно задает вещественное число вида

$$(-1)^s\ m\ 2^p,$$

где p — показатель экспонента, который представляет собой разность двоичного числа $p_{10} \ldots p_0$ и десятичного числа 1023 (смещение). Если двоичное число $p_{10} \ldots p_0$ отлично от нуля, то такие вещественные числа называются *нормализованными*, в противном случае — *денормализованными*. Мантисса m нормализованного числа определяется как двоичное число вида

$$1.\ m_{51}\ m_{50}\ \ldots\ m_1\ m_0,$$

а дерномализованного как

$$0.\ m_{51}\ m_{50}\ \ldots\ m_1\ m_0.$$

В случае если последовательность, определяющая показатель экспоненты $p_{10} \ldots p_0$, целиком состоит из единиц, а последовательность, задающая мантиссу $m_{51} \ldots m_0$, — целиком из нулей, то соответствующее число в зависимости от знака называется плюс бесконечностью (`+Inf`) или минус бесконечностью (`-Inf`). Если в мантиссе хотя бы один бит не нулевой, то соответствующее значение трактуется как не число (`Nan`).

Приведенных определений уже достаточно для более детального изучения (по сравнению с описанным во введении) машинной арифметики чисел с плавающей точкой на практических занятиях. Однако для этого необходимо предварительно сформировать вспомогательный программный инструментарий.

Вспомогательный программный инструментарий

Предполагается, что учащийся получит базовые знания по языку С, соответствующие, например, учебнику [2]. Однако в данном случае имеет смысл рассмотреть "расширенные" возможности, предоставляемые стандартом С99 [3]. Это касается типа `uint64_t`[1] (беззнаковых целых чисел размера 64 бита), определяемого в заголовочном файле `stdint.h`.

Основная идея подхода в изучении чисел, представимых в формате `double`, состоит в том, что один и тот же участок памяти размера 64 бита одновременно трактуется и как вещественное число, и как беззнаковое целое число. С последними уже можно работать при помощи стандартных битовых операций.

Первый способ реализации данного подхода базируется на работе с указателями. Определяется переменная типа `double`. Берется указатель (тип `double*`) на эту переменную, который преобразуется к указателю типа `uint64_t*`. Далее с содержимым этой переменной уже можно работать как с беззнаковым целым числом.

Другой подход (по мнению автора, методически более верный) базируется на использовании объединений (листинг 1). В соответствии с работой [2]: объединение — это переменная, которая может содержать объекты разных типов и размеров (но не одновременно); при этом удовлетворение к размеру и выравниванию возлагается на компилятор.

Листинг 1. Тип данных `numb_t`.

```
typedef union numb_t
{
    double r;
    uint64_t i;
} numb_t;
```

Если определить переменную `x` типа `numb_t`, то выражение `x.r` предоставляет доступ к значению переменной `x` как к вещественному числу, а выражение `x.i` — как к целому беззнаковому числу.

Заметим, что все представимые в формате `double` вещественные числа естественным образом занумерованы. При этом `x.i` является порядковым номером вещественного числа `x.r`. Для положительных вещественных чисел `x.r` и `y.r` из неравенства `x.i < y.i` следует неравенство `x.r < y.r`. Если `y.i = x.i + 1`, то `y.r` является следующим в порядке возрастания за `x.r` представимым вещественным числом.

Для анализа представимых вещественных чисел можно воспользоваться функцией `print`, приведенной в листинге 2. Данная функция печатает битовое представление, порядковый номер и само представимое вещественное число. Например, для числа `155.625` функция `print` напечатает

```
0 1000000011000110111010000000000000000000000000000000000000000000
4639679584470564864
1.55625000000000000000000000000000000000000000000000000000000000e+02.
```

Листинг 2. Функция `print`.

```
void print(numb_t n)
{
   int j;
   for(j = 63; j >= 0; j --)
   {
     printf(" %"PRIu64, (n.i >> j) & UINT64_C(1));
     if(j == 63 || j == 52)
        printf(" ");
   }
   printf("\n%"PRI64"\n%.55e\n", n.i, n.r);
}
```

В листинге 3 представлены функции `create`, `getP` и `getM`. Функция `create` формирует представимое ве-

[1] Встроенный тип unsigned long long должен иметь размер не менее 64 бит. В наших задачах нужно оперировать числами, размер которых строго равен 64 битам.

щественное число, используя в качестве входных аргументов число p, задающее последовательность бит показателя экспоненты, и число m, задающее последовательность бит мантиссы. В свою очередь, функции getP и getM выполняют обратные действия. Получая на вход представимое вещественное число, функция getP возвращает число, задающее последовательность бит показателя экспоненты, а getM — мантиссы.

Листинг 3. Функции `create`, `getP`, `getM`.

```
numb_t create(uint64_t p, uint64_t m)
{
      numb_t n;
      n.i = (p << 52) | m;
      return n;
}
uint64_t getP(numb_t n)
{
      return (n.i << 1) >> 53;
}
uint64_t getM(numb_t n)
{
      return (~UINT64_C(0) >> 12) & n.i;
}
```

Учебный практикум

Введенный вспомогательный программный инструментарий позволяет приступить к решению "разминочных" задач.

Задача 1. Найти наименьшее представимое положительное вещественное число.

Решение. Очевидно, что это будет денормализованное число, а следовательно, последовательность бит, задающая порядок экспоненты, должна состоять целиком из нулей. Младший бит мантиссы должен быть единичным, а остальные — нулевыми. Для формирования подобного числа можно воспользоваться следующим выражением:

```
create(UINT64_C(0), UINT64_C(1)).
```

Функция print выведет на экран следующую информацию:

```
0  00000000000000000000000000000000000000000000000000000000000000001
1
4.94065645841246544176568792868221372365059802614324764 43e-324.
```

Задача 2. Найти наибольшее денормализованное число.

Решение. Мантисса данного числа должна целиком состоять из единичных битов. Следовательно, для формирования подобного числа можно воспользоваться выражением

```
create(UINT64_C(0), ~UINT64_C(0) >> 12).
```

Функция print выведет на экран следующую информацию

```
0 00000000000 11111111111111111111111111111111111111111111111111
4503599627370495
2.22507385850720088902458687608585988765042311224095946 55e-308.
```

Задача 3. Проверить, является ли данное число *n* бесконечностью.

Решение. Можно воспользоваться введенными функциями getP и getM. У бесконечности мантисса целиком состоит из нулевых битов, а показатель экспоненты — из единичных. Следовательно, нужно проверить истинность выражения

```
getM(n) == UINT64_C(0) && getP(n) ==
      == ~(~UINT64_C(0) << 12).
```

Задача 4. Привести пример непредставимого в формате double вещественного числа.

Решение. Вызвав функцию print, можно убедиться, что число 0.1 не является представимым[2]:

```
0 01111111011 1001100110011001100110011001100110011001100110011010
4591870180066957722
1.0000000000000000055511151231257827021181583404541015625 0e-01
```

Задача 5. Привести пример возникновения погрешности при выполнении арифметических операций.

Решение. Результатом вычисления следующей суммы

```
0.01+ ... + 0.01 (десять раз)
```

будет число

```
0 01111111011 1001100110011001100110011001100110011001100110011001
4591870180066957721
9.99999999999999916733273153113259468227624893188476562 50e-02.
```

Данное число будет отличаться от машинного представления числа 0.1, приведенного выше.

Список подобных задач может быть продолжен.

Машинный нуль

Более интересной (и сложной) задачей является нахождение *машинного нуля*. Машинным нулем называется наибольшее представимое вещественное число $z > 0$, такое, что

$$1.0 + z == 1.0.$$

Естественно, в данном равенстве под знаком + понимается машинная реализация операции сложения.

При вычислении машинного нуля в начале находят его приближение. Для этого традиционно используют следующий алгоритм. Строится последовательность чисел

$$z_0, z_1, ..., z_n, ... (z_0 = 1.0, z_{i+1} = z_i/2.0).$$

Первое число z_n в этой последовательности, для которого выполняется равенство

$$1.0 + z_n == 1.0,$$

выбирается в качестве приближения машинного нуля. В листинге 4 приведен текст процедуры getZeroAppr, реализующей данный алгоритм.

[2] Для тестирования примеров программ использовался компилятор gcc 4.7.2 и операционная система Ubuntu 12.10 (32-битная и 64-битная версии).

Листинг 4. Функция `getZeroAppr`.

```
numb_t getZeroAppr(void)
{
   numb_t x, y, z;
   for(x.r = 1.0, y.r = 2.0, z.r = 2.0; x.i != y.i;
      z.r /= 2.0, y.r = x.r + z.r)
      ;
   return z;
}
```

Для 32-битной и 64-битной операционных систем в качестве приближения машинного нуля было получено следующее число:

```
0 01111001010 0000000000000000000000000000000000000000000000000000
4368491638549381120
1.1102230246251565404236316680908203125000000000000000000e-16.
```

Очевидно, что машинный нуль и его приближение будут иметь один и тот же показатель экспоненты, а отличаться будут только мантиссами. Используя вариант метода "деления пополам", можно найти требуемую мантиссу. Процедура `getZero`, текст которой приведен в листинге 5, решает данную задачу.

Листинг 5. Функция `getZero`.

```
   int checkZero(numb_t n)
      {
         numb_t x, y;
         x.r = 1.0;
         y.r = 1.0 + n.r;
         return (x.i == y.i) ? 1 : 0;
      }
      numb_t getZero(void)
      {
         uint64_t a, b, c, p;
         numb_t n;

         n = getZeroAppr();
         p = getP(n);

         for(a = UINT64_C(0), b = ~UINT64_C(0) >> 12; b - a > 1;)
         {

            c = a + ((b - a) >> 1);
            n = create(p, c);
            if(checkZero(n))
               a = c;
            else
               b = c;
         }
         n = create(p, a);
         return n;
      }
```

При проведении экспериментов оказалось, что для 64-битной операционной системы машинный нуль и его приближение совпадают. Для 32-битной операционной системы был получен следующий машинный нуль:

```
0 01111001010 0000000000100000000000000000000000000000000000000000
4368493837572636672
1.1107651257113992926406353944912552833557128906250000000e-16.
```

Заключение

В статье было представлено введение в машинную арифметику чисел с плавающей точкой. При написании статьи в качестве цели не ставилось подробное изложение данной предметной области. Многие важные темы, такие как округление, исключительные ситуации, возникающие при выполнении арифметических операций, формализация понятия единицы наименьшей точности, не были в достаточной мере затронуты. Необходимо отметить, что это вполне естественно, учитывая масштаб предметной области (один стандарт IEEE754 [4] занимает 58 страниц, а справочная книга [1] — 572 страницы).

В то же время с методической точки зрения статья носит завершенный характер. В ее основу положены материалы, которые использовались автором при проведении практических занятий по курсу "Работа на ЭВМ и программирование" для первоначального ознакомления учащихся с данной предметной областью.

Список литературы

1. **Muller J. M.** et al. Handbook of Floating-Point Arithmetic. Berlin: Springer, 2010. 572 p.

2. **Керниган Б., Ритчи Д.** Язык программирования С. М.: Вильямс, 2012. 304 с.

3. **ISO/IEC 9899:** TC2. Programming languages — C. ISO/IEC, 1999. 538 p.

4. **IEEE Std** 754—2008. IEEE Standard for Floating-Point Arithmetic. IEEE. 2008. 58 p.

CONTENTS

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

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

Сдано в набор 28.12.2012 г. Подписано в печать 13.02.2013 г. Формат 60×88 1/8. Заказ PI313
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.