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 004.432.2

Н. И. Вьюкова, стар. науч. сотр., **В. А. Галатенко**, д-р физ.-мат. наук, стар. науч. сотр.,
С. В. Самборский, стар. науч. сотр., НИИ системных исследований РАН, г. Москва,
e-mail: niva@niisi.msk.ru

Модель памяти многопоточных программ в стандарте C11

Представлены библиотека атомарных операций языка C11, модель памяти многопоточной программы и механизмы синхронизации потоков при помощи атомарных операций. Также рассмотрена поддержка атомарных операций языка C в текущих версиях компиляторов GCC и Clang/LLVM.

***Ключевые слова**: язык программирования C, C11, многопоточность, атомарные типы данных, модель памяти*

Введение

Настоящая статья продолжает серию статей [1, 2], посвященных стандарту языка C, принятому в декабре 2011 г. и получившему неформальное название C11 [3]. Общий обзор нововведений, вошедших в C11, представлен в работе [1].

Одним из наиболее значимых новшеств C11 является поддержка многопоточных программ, которая включает ряд языковых и библиотечных средств и опирается на несколько новых понятий и определений. Этой теме посвящена статья [2], где были рассмотрены базовые средства разработки многопоточных программ, а также настоящая статья, в которой представлена модель памяти многопоточных программ.

Стандарт C11 (см. [3], п. 5.1.2.4) явно вводит понятие многопоточной программы и определяет выполнение программы как выполнение совокупности ее потоков. Два основных инструментария разработки многопоточных приложений — библиотека управления потоками **<threads.h>** и библиотека для работы с атомарными объектами **<stdatomic.h>**. Библиотека **<threads.h>** предоставляет функции для работы с потоками, мьютексами, условными переменными, индивидуальными данными потоков. Библиотека **<stdatomic.h>** содержит определения стандартных атомарных типов, функции инициализации атомарных объектов и набор операций над ними. Обе эти библиотеки относятся к числу выборочных (необязательных) возможностей.

В работе [2] рассмотрена библиотека **<threads.h>**, а также понятие атомарного типа данных и базовая часть библиотеки **<stdatomic.h>**, включая стандартные атомарные типы данных языка C11, свойство неблокируемости операций над атомарными объектами и атомарные операции, позволяющие организовать выполнение многопоточной программы в режиме последовательной консистентности. Этот режим характеризуется тем, что работа многопоточной программы может трактоваться как перемежающееся последовательное выполнение вычислений из составляющих ее потоков.

Режим последовательной консистентности обеспечивает надежность и простоту понимания многопоточных программ, но его реализация, как правило, требует значительных накладных расходов (см. [2], подразд. 3.7, пример 2). По этой причине в стандарте C11 предусмотрены другие, менее строгие механизмы синхронизации, а также атомарные операции, не участвующие в синхронизации, что позволяет обеспечить более эффективную реализацию многопоточных программ.

Предметом рассмотрения настоящей статьи является модель памяти многопоточных программ, включая такие ее аспекты, как упорядочение вычислений в многопоточной программе, правила модификации объектов в памяти, гонка данных. Также рассмотрены различные механизмы синхронизации, предоставляемые библиотекой **<stdatomic.h>**, и некоторые вопросы генерации кода, связанные с реализацией атомарных операций.

1. Модель памяти

Одним из центральных понятий модели памяти языка C11 является межпоточное отношение "раньше", которое устанавливает частичный порядок на множестве вычислений, проводимых в разных потоках программы. Это позволяет программисту требуемым образом упорядочивать операции над объектами в общей памяти. Отношение "раньше" устанавливается при

помощи определенных "действий над памятью", которые выполняются атомарными операциями, а также операциями захвата и освобождения мьютекса.

Гонка данных (которая в C11 трактуется как неопределенное поведение) возникает в ситуации, когда операции над неатомарным объектом, проводимые в разных потоках, неупорядочены, притом что по крайней мере одна из этих операций является записью. Заметим, что неупорядоченность операций над атомарным объектом не является источником гонки данных. В частности, стандарт явно вводит класс атомарных операций, не участвующих в синхронизации.

Еще одно важное понятие, связанное с моделью памяти, — порядок модификаций объекта в памяти, т. е. правила, относящиеся к последовательности значений, принимаемых объектом в ходе выполнения программы.

Все эти понятия и определения рассмотрены далее в этом разделе. В конце раздела приведены примеры, иллюстрирующие различные методы синхронизации и особенности атомарных операций, не участвующих в синхронизации.

1.1. Атомарные операции с явным заданием упорядочивающих действий

Стандарт C11, как и прежние стандарты языка C, определяет отношение следования между вычислениями внутри одного потока, основанное на понятиях точек следования (*sequence points*). Сведения о них можно найти в Приложении C (в работе [3]). Прежние стандарты требовали, чтобы видимое, внешнее поведение программы было таким, как если бы порядок выполнения вычислений соответствовал отношению следования.

Отмеченные выше соображения указывают на то, что должен соблюдаться порядок операций с внешними устройствами. В частности, не должен изменяться порядок операций над объектами, объявленными как **`volatile`**, поскольку они могут быть связаны с внешними устройствами. В остальном же компиляторы могли существенно переупорядочивать вычисления, а также удалять или добавлять операции чтения-записи объектов в целях оптимизации. В многопоточной программе изменения объектов в общей памяти, сделанные одним потоком, видны другим потокам, следовательно, приведенных выше понятий об упорядоченности вычислений и о видимом поведении программы недостаточно. Примеры, иллюстрирующие ошибки, которые могут возникать при использовании языка C для создания многопоточных программ, можно найти в работе [2], разд. 3.7 и разд. 5.

В стандарте C11 вводится ряд новых понятий, описывающих взаимодействие между потоками, в частности, понятие межпоточного отношения "раньше". В этом разделе будут представлены свойства атомарных операций, на которых основано отношение "раньше".

С атомарной операцией может быть связано синхронизирующее действие над памятью: *release, acquire, consume* или *acq_rel* (*acquire* и *release*). Говоря неформально, действие *release* над объектом *M* завершает и делает видимыми все выполненные ранее побочные эффекты вычислений другому потоку, выполнившему над объектом *M* действие *acquire* или *consume*.

Помимо атомарных функций, рассмотренных в работе [2], библиотека **`<stdatomic.h>`** содержит набор аналогичных функций, имеющих дополнительный аргумент перечислимого типа **`memory_order`**. Имена этих функций имеют суффикс **`_explicit`**, а дополнительный аргумент задает действие над памятью.

Приведем возможные значения перечислимого типа **`memory_order:`**

- **`memory_order_relaxed`** — указывает, что данная операция не участвует в синхронизации;
- **`memory_order_consume`** — соответствует действию *consume* для операции чтения, не допускается для операций записи;
- **`memory_order_acquire`** — соответствует действию *acquire* для операции чтения, не допускается для операций записи;
- **`memory_order_release`** — соответствует действию *release* для операции записи, не допускается для операций чтения;
- **`memory_order_acq_rel`** — допускается только для операций чтения-модификации-записи (*Read-Modify-Write* — RMW), соответствует *acquire* для чтения и *release* для записи;
- **`memory_order_seq_cst`** — допускается для всех операций, соответствует *acquire* для чтения и *release* для записи, но задает более строгий режим последовательной консистентности, это значение подразумевается для атомарных операций, описанных в работе [2], разд. 3.4.

Ниже перечислены операции библиотеки **`<stdatomic.h>`** с явным заданием синхронизирующего действия. Семантика этих операций совпадает с семантикой соответствующих операций, описанных в работе [2], за исключением того, что в них при помощи дополнительного аргумента явно задается действие над памятью.

```
void atomic_store_explicit(volatile A *object,
   C desired, memory_order order);
C atomic_load_explicit(volatile A *object,
   memory_order order);
C atomic_exchange_explicit(volatile A *object,
   C desired, memory_order order);
_Bool atomic_compare_exchange_strong_explicit(
volatile A *object, C *expected, C desired,
memory_order success, memory_order failure);
_Bool atomic_compare_exchange_weak_explicit(
   volatile A *object, C *expected, C desired,
   memory_order success, memory_order failure);
C atomic_fetch_key_explicit(volatile A *object,
   M operand, memory_order order);
_Bool atomic_flag_test_and_set_explicit(
   volatile atomic_flag *object, memory_order
      order);
void atomic_flag_clear_explicit(
   volatile atomic_flag *object, memory_order
      order);
```

Синхронизирующее действие может также быть выполнено без указания конкретного объекта в памяти при помощи операций, называемых барьерами памяти (*fence*):

```
void atomic_thread_fence (memory_order order);
order == memory_order_relaxed // барьер игно-
//рируется;
order == memory_order_acquire // или
memory_order_consume // барьер типа aquire;
order == memory_order_acq_rel // барьер типа
//acquire/release;
order == memory_order_seq_cst // последовательно-
//консистентный барьер типа acquire/release.
void atomic_signal_fence(memory_order order);
```

Функция `atomic_signal_fence` действует так же, как `atomic_thread_fence`, но порядок операций устанавливается только по отношению к обработчику сигналов, выполняемому внутри того же потока. Такой барьер влияет на упорядочение операций компилятором, но не требует вставки дополнительных команд синхронизации, например, для выталкивания данных из внутренних буферов процессора в разделяемую память.

1.2. Упорядочение операций над памятью в многопоточной программе

На основе синхронизирующих действий, выполняемых атомарными операциями и операциями над мьютексами, стандарт определяет отношение порядка "раньше" (*happens before*) между вычислениями, выполняемыми в разных потоках, подобно тому, как в модели Лампорта [4] вводится частичное упорядочение событий в процессах, обменивающихся сообщениями.

Отношение "раньше" включает ряд вспомогательных отношений порядка, которые рассмотрены далее.

Отношение *внутрипоточного следования* (*sequenced before*) — транзитивное отношение следования между вычислениями внутри потока, основанное на понятии точек следования (см. [3], п. 5.1.2.3).

Определение 1. Отношение *зависимости по данным* — подмножество отношения внутрипоточного следования, транзитивное замыкание обычного отношения зависимостей[1], включая зависимости по записи-чтению объекта в памяти.

Определение 2. Отношение *синхронизации* (межпоточное) — имеет место между атомарными операциями *A* и *B*, если *A* выполняет запись объекта *M* с действием *release*, а *B* в другом потоке выполняет чтение объекта *M* с действием *acquire* и считывает значение, записанное операцией *A*, или одно из значений *release*-цепочки операции *A*.

Отношение синхронизации может возникать как между операциями библиотеки `<stdatomic.h>`, так и между библиотечными вызовами, реализующими освобождение мьютекса и его последующий захват. Считается, что операция освобождения мьютекса осуществляет запись с действием *release* в объект, описывающий мьютекс, а операция захвата мьютекса осуществляет чтение этого объекта с действием *acquire*. Все операции, выполняемые в программе над конкретным мьютексом, упорядочены.

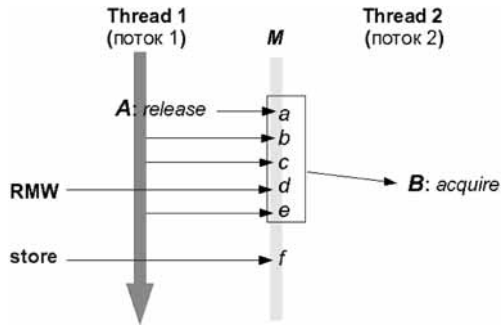


Рис. 1. ***Release*-цепочка операции *A*. Цепочка завершается, когда другой поток совершает запись (не RMW) в объект *M***

Определение 3. *Release*-цепочка атомарной операции *A*, выполняющей запись объекта *M*, — это максимальная непрерывная подпоследовательность побочных эффектов в последовательности модификаций *M*, начинающаяся с *A*, и такая, что все последующие операции в ней либо принадлежат тому же потоку, либо являются RMW-операциями (рис. 1). Таким образом, *release*-цепочка завершается, когда другой поток выполняет запись (не RMW) в объект *M*.

Особая роль RMW-операций в процессе синхронизации заключается в том, что они не прерывают *release*-цепочку, сохраняя возможность синхронизации между выполненной ранее записью (*release*) и последующим чтением (*acquire* или *consume*) в другом потоке.

Определение 4. Отношение *предшествования по зависимостям* (межпоточное) имеет место между атомарными операциями *A* и *B*, если:

- *A* выполняет запись объекта *M* с действием *release*, а *B* в другом потоке выполняет чтение объекта *M* с действием *consume* и считывает значение, записанное операцией *A* или одно из значений из *release*-цепочки операции *A*[2];
- или существует вычисление *X*, такое что *A* предшествует *X* по зависимостям, а *B* зависит по данным от *X*.

Определение 5. Отношение *"межпоточно раньше"* (*inter-thread happens before*) имеет место между вычислениями *A* и *B*, если:

- *A* и *B* синхронизированы;
- или *A* предшествует *B* по зависимостям;
- или существует такое вычисление *X*, что
  - между *A* и *X* имеет место синхронизация, а между *X* и *B* — отношение следования;
  - или *A* и *X* связаны отношением следования и *X* "межпоточно раньше" *B*,
  - или *A* "межпоточно раньше" *X* и *X* "межпоточно раньше" *B*.

Таким образом, отношение "межпоточно раньше" описывается как подмножество транзитивного замы-

[1] С некоторыми исключениями, на которых здесь не останавливаемся.

[2] То же, что синхронизация, но *B* выполняет не *acquire*, а *consume*; поэтому далее это отношение будем называть также синхронизацией *release-consume*.

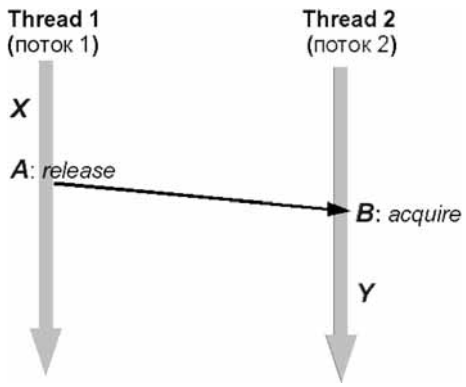


Рис. 2. Отношение «раньше» с синхронизацией *release-acquire*

кания от объединения отношений синхронизации, внутрипоточного следования и предшествования по зависимостям. Вычисление A "межпоточно раньше" вычисления A', если существует цепочка вычислений $A_0, A_1, ..., A_n$, где $A_0 = A$ и $A_n = A'$ и каждая пара A_i, A_{i+1} в ней связана одним из перечисленных выше отношений со следующими двумя исключениями.

Цепочка не должна завершаться предшествованием по зависимостям с последующим внутрипоточным следованием. Ограничение введено потому, что операция *consume*, участвующая в предшествовании по зависимости, подразумевает дальнейшее упорядочение только тех операций (внутри данного потока), которые зависят по данным от *consume*. Считается, что последующая операция *release* установит порядок для операции *consume*.

Цепочка не должна состоять только из отношений внутрипоточного следования.

Определение 6. Отношение "раньше" имеет место между вычислениями A и B, если A следует перед B или A "межпоточно раньше" B.

Рис. 2 иллюстрирует отношение "раньше" с синхронизацией *release-acquire*. Здесь между X и A, а также между B и Y имеет место отношение следования, а A и B связаны отношением синхронизации, соответственно, X "раньше" A "раньше" B "раньше" Y.

Рис. 3 иллюстрирует отношение предшествования по данным, являющееся частным случаем отношения "раньше". Здесь A и B связаны отношением синхронизации *release-consume*, а также между B и $Y1$, $Y2$, $Y3$ имеет место отношение зависимости по данным.

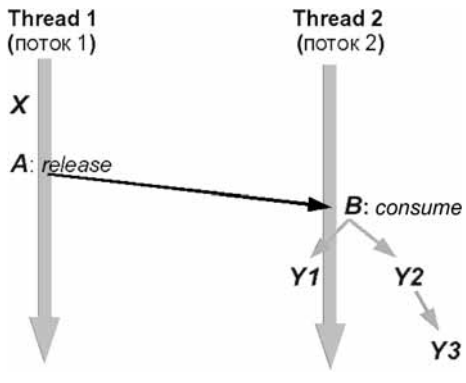


Рис. 3. Отношение «раньше» с синхронизацией *release-consume*

Из определения отношения "раньше" вытекает ряд требований для компилятора:

- операции над памятью, следующие перед атомарной *release*-операцией A, не могут быть перенесены "вниз", за A;
- операции над памятью, следующие после атомарной *acquire*-операции B, не могут быть перенесены "вверх", за B;
- операции над памятью, следующие после атомарной *consume*-операции B и зависящие от нее по данным, не могут быть перенесены "вверх", за B.

Если целевой процессор переупорядочивает инструкции, то может потребоваться вставка инструкции, запрещающей перенос операций чтения-записи. Если в многопроцессорной конфигурации процессоры имеют буферы памяти на запись, то может потребоваться вставка инструкции для синхронизации буфера с общей памятью (см. примеры в подразд. 1.5).

Синхронизация *release-consume* оставляет компилятору и, что важнее, процессору больше возможностей для переупорядочения вычислений, чем *release-acquire*. Большинство современных процессоров обеспечивает упорядочение, которое необходимо для синхронизации *release-consume*, без вставки дополнительных инструкций.

Важно понимать, что для атомарных переменных, не являющихся `volatile`, стандарт специфицирует только видимое поведение. Это означает, что компилятор может переупорядочивать операции над атомарными переменными, если он сумел доказать те или иные свойства программы. Например, если некоторая атомарная переменная используется только в одном потоке, то компилятор может произвольно оптимизировать операции с этой переменной. Или, если поток 1 пишет в переменные A и B, поток 2 читает только A, поток 3 читает только B и потоки 2, 3 не взаимодействуют между собой, то компилятор может переупорядочивать записи в A и B.

В целом C11 предоставляет три базовых механизма синхронизации — атомарные операции (выполнение *release* и последующее выполнение *acquire* или *consume*), барьеры памяти и мьютексы. Заметим, что захват и освобождение мьютекса `m`, согласно стандарту, включает выполнение действий, соответственно, *acquire* и *release* с объектом `m`.

Отношения синхронизации с участием барьеров описывают следующие определения.

Определение 7. Между *release*-барьером A и *acquire*-барьером B имеет место отношение синхронизации, если существуют атомарные операции X, Y такие, что

- A следует перед X и X записывает объект M;
- Y следует перед B и Y читает значение, записанное X, или значение из *release*-цепочки X[3].

(См. пример 2 в разд. 1.5 далее).

[3] В определениях 7, 8 X — не обязательно *release*-операция. В частности, X может быть *relaxed*-операцией. В этих двух определениях имеется в виду гипотетическая *release*-цепочка, берущая начало от X, как если бы X была *release*-операцией.

Определение 8. Между *release*-барьером *A* и атомарной *aquire*-операцией *Y*, читающей объект *M*, имеет место отношение синхронизации, если существует атомарная операция *X* такая, что

- *A* следует перед *X* и *X* записывает объект *M*;
- *Y* читает значение, записанное *X,* или значение из *release*-цепочки *X*.

Определение 9. Между атомарной *release*-операцией *X*, записывающей объект *M*, и *acquire*-барьером *B* имеет место отношение синхронизации, если существует атомарная операция *Y* такая, что *Y* следует перед *B* и *Y* читает значение, записанное *X,* или значение из *release*-цепочки *X*.

1.3. Понятие гонки данных

Следующие два определения вводят понятие гонки данных (*data race*).

Определение 10. Два вычисления конфликтуют, если одно из них модифицирует некоторую позицию памяти, а другое читает или модифицирует эту же позицию в памяти.

Определение 11. Гонка данных имеет место при выполнении программы, если она содержит два конфликтующих вычисления в разных потоках, по крайней мере одно из которых не атомарно, и они не связаны отношением "раньше". В таком случае поведение программы неопределенно.

Согласно этому определению, *relaxed*-операции, хотя и не участвуют в синхронизации, не могут быть источником гонки данных, поскольку являются атомарными. Заметим, что инициализация атомарных объектов (см. [2], подразд. 3.3) не является атомарной операцией. Если доступ к объекту выполняется одновременно с его инициализацией, то имеет место гонка данных.

Пример программы, содержащей гонку данных, представлен в работе [2], подразд. 3.7 (пример 1). В этой программе гонка данных имеет место по обеим переменным `flag` и `data`. Другие часто встречающиеся виды гонки данных в многопоточных программах представлены в работе [5].

Следует иметь в виду, что компилятор, принимая решения о возможности тех или иных оптимизаций, исходит из того, что в программе отсутствует гонка данных. Полезное обсуждение практики компиляции и ситуаций, когда оптимизация приводит к ошибкам в силу присутствия гонки данных, можно найти в публикации [6].

1.4. Последовательность модификаций объекта в памяти

Стандарт также вводит понятия *последовательности модификаций* (*modification order*) атомарного объекта *M*. Если операции *A* и *B* модифицируют объект *M*, и *A* "раньше" *B*, то результат *A* следует раньше результата *B* в последовательности модификаций объекта *M*.

Заметим, что речь идет об отдельных последовательностях для каждого атомарного объекта. Нет никаких требований, что из этих последовательностей может быть выведена некоторая общая последовательность модификаций всех объектов. В общем случае это невозможно, поскольку различные потоки могут наблюдать модификации объектов в разном порядке.

Компилятор не должен переупорядочивать атомарные операции над одним и тем же объектом, включая операции чтения, даже если это *relaxed*-операции. Это относится и к ситуациям, когда нет возможности доказать, что объекты, к которым осуществляется доступ, разные. Например, пусть некоторый поток периодически выполняет *relaxed*-считывание объекта *M* для того чтобы отображать его значения графически. Данное требование обеспечивает верное отображение динамики изменений — если последовательность значений, принимаемых объектом *M*, монотонно возрастает, то и последовательность считываемых значений будет монотонно возрастающей.

Еще одно требование стандарта, имеющее важные следствия для реализации компиляторов, заключается оно в том, что каждое значение в последовательности модификаций объекта должно быть результатом какого-либо вычисления, явно присутствующего в программе. Это означает, что некоторые широко применяемые в практике компиляции методы генерации кода и оптимизации, которые порождают дополнительные записи в разделяемую память (см. примеры в работе [2], разд. 5), должны быть исключены.

Что касается методов, использующих вставку спекулятивных считываний из разделяемой памяти, то в реализациях для большинства целевых машин такие методы не приведут к видимому изменению в поведении программы. Однако они могут оказаться некорректными в компиляторах для (гипотетических) машин с автоматическим определением гонки данных.

1.5. Примеры использования различных типов синхронизации

Рассмотрим ряд примеров, иллюстрирующих особенности различных типов синхронизации. В работе [2] (см. разд. 3.7, пример 2) представлена программа, в которой используется синхронизация в режиме последовательной консистентности. Ниже в примере 1 показана реализация того же алгоритма при помощи синхронизации *release-acquire*.

Пример 1. Использование синхронизации *release-acquire*.

```
int data = 0;
atomic_int flag = ATOMIC_VAR_INIT (0);
// Thread 1:
// запись данных
data = 123;                  //X
atomic_store_explicit (&flag, 1,
  memory_order_release)     //A

// Thread 2:
if (atomic_load_explicit (&flag,
  memory_order_acquire))    //B
// обработка данных:
printf ("%d", data);         //Y
```

Здесь комментарии *X*, *A*, *B*, *Y* соответствуют операциям, показанным на рис. 2 в подразд. 1.2. В данном примере действительно достаточно использовать синхронизацию *release-acquire*. Если второй поток прочитал значение `flag == 1`, записанное первым

потоком, то имеет место синхронизация между операциями **atomic_store** и **atomic_load**, а значит, по определению, между операциями записи в **data** и чтения из **data** в двух потоках существует отношение "раньше".

В примере 2 показана реализация этого алгоритма при помощи барьеров. Комментарии *A*, *X*, *Y*, *B* соответствуют операциям из определения 7 в подразд. 1.2.

Пример 2. Синхронизация *release-acquire* при помощи барьеров.

```
int data = 0, f = 0;
atomic_int flag = ATOMIC_VAR_INIT (0);
// Thread 1:
data = 123; // запись данных
atomic_thread_fence(memory_order_release) //A
atomic_store_explicit
   (&flag, 1, memory_order_relaxed)     //X

// Thread 2:
f =(atomic_load_explicit
(&flag, memory_order_relaxed))            //Y
atomic_thread_fence(memory_order_acquire); //B
// обработка данных
if (f) printf ("%d", data);
```

Важно понимать, что режим последовательной консистентности не всегда можно заменить синхронизацией *release-acquire*. В примере 3 представлена программа, которая может вести себя по-разному в этих двух режимах. Предполагается, что начальные значения переменных **x**, **y** равны **0**.

Пример 3. Различие между последовательной консистентностью и синхронизацией *release-acquire*.

```
// Thread 1:
atomic_store_explicit
   (&x, 1, memory_order_release)
r1 = atomic_load_ explicit
   (&y, memory_order_acquire)

// Thread 2:
atomic_store_explicit
   (&y, 1, memory_order_release)
r2 = atomic_load_explicit
   (&x, memory_order_acquire)
```

Для данной программы возможен результат **r1 == 0, r2 == 0**. Это может произойти, например, если в потоке 1 при выполнении **store** в переменную **x** значение 1 осталось в буфере записи процессора и не было вытолкнуто в общую память до выполнения **load** во втором потоке (аналогичная ситуация может произойти при записи и чтении **y**). Отношение синхронизации определяется стандартом условно: "если операция *acquire* прочитала значение, записанное операцией *release* в другом потоке...". Если же значение (по каким-то причинам) не было прочитано, то синхронизации нет, а следовательно нет и отношения упорядоченности между операциями двух потоков.

Однако, если бы в рассматриваемой программе был использован режим последовательной консистентности, то результат **r1 == 0, r2 == 0** был бы невозможен, поскольку он не может иметь место ни при каком варианте перемежающегося последовательного выполнения потоков.

С теоретической точки зрения разница между режимами последовательной консистентности и *release-acquire* состоит в том, что согласно стандарту должен существовать общий для всей программы (одинаково видимый всеми потоками) порядок выполнения атомарных операций *seq_cst*. Для других операций это не так, разные потоки могут видеть порядок их выполнения по-разному, что иллюстрирует пример 3.

С точки зрения генерации кода, различие заключается в том, что после операции **store** в режиме последовательной консистентности компилятор должен вставить команду, обеспечивающую выталкивание данных из внутреннего буфера в общую память (если такие буферы существуют). Это показывает, что использование последовательной консистентности может приводить к снижению эффективности программ по сравнению с другими способами синхронизации.

В примере 4 представлена программа, в которой достаточно синхронизации *release-consume*. Обычно данный тип синхронизации применяется при чтении указателей с последующим доступом к данным по этому указателю или к массиву с использованием индексов.

Пример 4. Использование синхронизации *release-consume*.

```
int data = 0, *p;
atomic_intptr_t pdata = ATOMIC_VAR_INIT (0);

// Thread 1:
data = 123; // запись данных
atomic_store_explicit (&pdata,
   (intptr_t) &data, memory_order_release);

// Thread 2:
// проверка флага
if (p = (int *) atomic_load_explicit
   (&pdata, memory_order_ consume))
// обработка данных
printf ("%d", *p);
```

Здесь во втором потоке операция доступа к ***p** в вызове **printf** зависит по данным от значения, считанного в первой строке; поэтому синхронизация *release-consume* достаточна для обеспечения правильной упорядоченности вычислений.

Следующие примеры иллюстрируют поведение атомарных операций, использующих **memory_order_relaxed**.

Пример 5. Атомарные операции *relaxed*.

```
// Thread 1:
r1 = atomic_load_explicit
   (&y, memory_order_relaxed);
atomic_store_explicit
   (&x, r1, memory_order_relaxed);
```

```
// Thread 2:
r2 = atomic_load_explicit
   (&x, memory_order_relaxed);
atomic_store_explicit
   (&y, 42, memory_order_relaxed);
```

В этом примере, при начальных значениях `r1 == 0, r2 == 0`, возможен результат `r1 == 42, r2 == 42`. Порядок вычислений, при котором достигается этот результат, следующий:

```
atomic_store_explicit
   (&y, 42, memory_order_relaxed);
r1 = atomic_load_explicit
   (&y, memory_order_relaxed);
atomic_store_explicit
   (&x, r1, memory_order_relaxed);
r2 = atomic_load_explicit
   (&x, memory_order_relaxed);
```

Такой порядок возможен, поскольку операции потока 2 независимы и могут быть переупорядочены.

Пример 6. Атомарные операции *relaxed*.

```
// Thread 1:
r1 = atomic_load_explicit
   (&y, memory_order_relaxed);
atomic_store_explicit
   (&x, r1, memory_order_relaxed);

// Thread 2:
r2=atomic_load_explicit
   (&x, memory_order_relaxed);
atomic_store_explicit
   (&y, r2, memory_order_relaxed);
```

В примере 6 результат `r1 == 42, r2 == 42` невозможен, так как ни одна последовательность вычислений в этой программе не приводит к вычислению значения 42.

Пример 7. *Relaxed*-операции — рекомендуемая практика компиляции.

В программе

```
// Thread 1:
r1 = atomic_load_explicit
   (&x, memory_order_relaxed);
if (r1 == 42)
atomic_store_explicit
   (&y, r1, memory_order_relaxed);

// Thread 2:
r2 = atomic_load_explicit
   (&y, memory_order_relaxed);
if (r2 == 42)
atomic_store_explicit
   (&x, 42, memory_order_relaxed);
```

результат `r1 == 42 && r2 == 42` формально возможен[4]. Тем не менее в стандарте дается рекомендация избегать подобной практики в реализациях компиляторов, поскольку такое поведение вряд ли можно считать полезным.

[4] То есть, основываясь на спецификациях стандарта, нельзя доказать, что данный результат невозможен.

2. Поддержка атомарных операций в компиляторах GCC и Clang/LLVM

В настоящее время свободно-распространяемые компиляторы GCC [7] и Clang-LLVM [8, 9] не поддерживают атомарные операции на уровне синтаксиса входного языка и стандартных библиотек. Тем не менее оба компилятора поддерживают встроенные (*builtin*) функции, реализующие атомарные операции.

Встроенные функции Clang-LLVM:

```
__c11_atomic_init
__c11_atomic_thread_fence
__c11_atomic_signal_fence
__c11_atomic_is_lock_free
__c11_atomic_store
__c11_atomic_load
__c11_atomic_exchange
__c11_atomic_compare_exchange_strong
__c11_atomic_compare_exchange_weak
__c11_atomic_fetch_add
__c11_atomic_fetch_sub
__c11_atomic_fetch_and
__c11_atomic_fetch_or
__c11_atomic_fetch_xor
```

Clang поддерживает также встроенные функции GCC, реализующие атомарные операции.

Ниже приведен список встроенных функций GCC. Для атомарных операций *load*, *store*, *exchange* и *compare_exchange* помимо аналогов стандартных функций GCC поддерживает обобщенные функции, позволяющие работать с данными произвольных типов. Интерфейс обобщенных функций отличается тем, что все аргументы передаются по указателю, и результат также передается по указателю.

В отличие от функций стандарта C11, встроенные функции GCC поддерживают неконстантные значения аргумента, задающего действие над памятью (`memmodel`). Для неконстантных значений при компиляции подразумевается самое сильное значение `memory_order_seq_cst` (последовательная консистентность).

```
TYPE __atomic_load_n (TYPE *ptr, int
memmodel)
```

Эта функция реализует `atomic_load_explicit`.

```
void __atomic_load (TYPE *ptr, TYPE *ret,
int memmodel)
```

Обобщенный вариант `atomic_load_explicit`.

```
void __atomic_store_n (TYPE *ptr, TYPE val,
int memmodel)
```

Эта функция реализует `atomic_store_explicit`.

```
void __atomic_store (TYPE *ptr, TYPE *val,
  int memmodel)
```

Обобщенный вариант `atomic_store_explicit`.

```
TYPE __atomic_exchange_n (TYPE *ptr, TYPE val,
  int memmodel)
```

Эта функция реализует `atomic_exchange_explicit`.

```
void __atomic_exchange (TYPE *ptr, TYPE *val,
  TYPE *ret, int memmodel)
```

Обобщенный вариант **atomic_exchange_explicit**.

```
bool __atomic_compare_exchange_n (TYPE *ptr,
  TYPE *expected, TYPE desired, bool weak, int
  success_memmodel, int failure_memmodel)
```

В зависимости от значения аргумента **weak** эта функция реализует **atomic_compare_exchange_ strong_ explicit** или **atomic_compare_exchange_weak_ explicit**.

```
bool __atomic_compare_exchange (TYPE *ptr,
  TYPE*expected, TYPE *desired, bool weak,
  int success_memmodel, int failure_memmodel)
```

Обобщенный вариант **atomic_compare_exchange_ strong_ explicit** или **atomic_compare_exchange_ weak_explicit**.

```
TYPE __atomic_add_fetch (TYPE *ptr, TYPE val,
   int memmodel)
TYPE __atomic_sub_fetch (TYPE *ptr, TYPE val,
   int memmodel)
TYPE __atomic_and_fetch (TYPE *ptr, TYPE val,
   int memmodel)
TYPE __atomic_xor_fetch (TYPE *ptr, TYPE val,
   int memmodel)
TYPE __atomic_or_fetch (TYPE *ptr, TYPE val,
   int memmodel)
TYPE __atomic_nand_fetch (TYPE *ptr, TYPE val,
   int memmodel)
```

Эти функции реализуют **atomic_fetch_***.

Компиляторы LLVM и GCC поддерживают локальные переменные потоков, хотя в GCC эта возможность реализована не для всех целевых платформ. Заметим, что реализация этого средства требует существенной поддержки со стороны операционной системы.

Clang обеспечивает углубленный анализ многопоточных программ с выдачей диагностики о возможных проблемах синхронизации доступа к объектам в разделяемой памяти [10]. Для управления анализом программист должен использовать атрибуты, указывающие свойства переменных, например, атрибут **guarded_var** указывает, что доступ к переменной должен быть защищен блокировкой.

Инструментальное средство ThreadSanitizer [5], развивающееся как в Clang [11], так и в GCC [12], позволяет осуществлять динамический анализ программ в целях выявления гонки данных во время выполнения.

Заключение

В стандарте C11 впервые введено понятие многопоточной программы и специфицированы три механизма синхронизации: мьютексы, атомарные операции и барьеры памяти. Реализация синхронизирующих действий может требовать значительных ограничений по оптимизации кода. По этой причине стандарт предоставляет несколько режимов синхронизации: последовательная консистентность, *release-acquire*, *release-consume*, а также атомарные операции, не участвующие в синхронизации. Это обеспечивает гибкие возможности для организации межпоточного взаимодействия с минимальными потерями производительности.

Стандарт вводит атомарные типы данных и определяет набор стандартных атомарных типов, соответствующих целочисленным типам языка C. При помощи макросов программа может опросить свойство блокируемости стандартных атомарных типов. Это позволяет программисту выбрать типы данных, обеспечивающие наиболее эффективное выполнение программы.

В стандарте определены понятия, описывающие модель памяти многопоточных программ, включая понятия межпоточного отношения "раньше", порядка модификации объектов в памяти, гонки данных. Строгое определение модели памяти дает хорошую основу как для создания многопоточных программ на языке C, так и для разработки компиляторов и средств анализа многопоточных программ.

В настоящее время в компиляторах GCC и Clang/LLVM на уровне синтаксиса входного языка отсутствует поддержка средств управления потоками, а также атомарных типов и операций, хотя семантика атомарных операций реализована на уровне встроенных функций. Напомним, что средства управления потоками и поддержка атомарных типов данных являются выборочными (необязательными) возможностями C11. Тем, кто хотел бы использовать возможности C11, не дожидаясь их реализации в доступных компиляторах, можно порекомендовать пакет заголовочных файлов P99 [13], который позволяет эмулировать ряд средств C11, включая библиотеки **<stdatomic.h>** и **<threads.h>**, при помощи компилятора GCC и библиотеки **<pthread.h>**.

Авторы выражают признательность Дмитрию Вьюкову (Google Inc., сайт http://www.1024cores.net) за полезные и содержательные консультации по вопросам, связанным с моделью памяти и атомарными операциями, а также за многочисленные ценные замечания по текстам данной статьи и работ [1, 2].

Список литературы

1. **Вьюкова Н. И., Галатенко В. А., Самборский С. В.** О стандарте C11 // Программная инженерия. 2013. № 8. С. 2—9.

2. **Вьюкова Н. И., Галатенко В. А., Самборский С. В.** Поддержка многопоточности в стандарте C11 // Программная инженерия. 2013. № 9.

3. **ISO/IEC 9899:2011** — Information technology — Programming Languages.

4. **Lamport L.** Time, Clocks, and the Ordering of Events in a Distributed System // Communications of the ACM. 1978. July. Vol. 21, N 7. P. 558—565.

5. **ThreadSanitizer**, a data race detector for C/C++ and Go. URL: http://code.google.com/p/thread-sanitizer.

6. **Benign** data races: what could possibly go wrong? URL: http://software.intel.com/en-us/blogs/2013/01/06/benign-data-races-what-could-possibly-go-wrong.

7. **GCC**, the GNU Compiler Collection. URL: http://gcc.gnu.org.

8. **Clang:** a C language family frontend for LLVM. URL: http://clang.llvm.org.

9. **The** LLVM Compiler Infrastructure. URL: http://llvm.org.

10. **Clang** Language Extensions. Extensions for Static Analysis. URL: http://clang.llvm.org/docs/LanguageExtensions.html#thread-safety-annotation-checking.

11. **Clang** 3.3 Documentation. ThreadSanitizer. URL: http://clang.llvm.org/docs/ThreadSanitizer.html.

12. **Using** AddressSanitizer & ThreadSanitizer In GCC 4.8. URL: http://www.phoronix.com/scan.php?page=news_item&px=MTIzOTU.

13. **P99** — Preprocessor macros and functions for C99. URL: http://p99.gforge.inria.fr/

УДК 004.4`24

Д. С. Одякова, вед. инж., **Р. В. Парахин**, аспирант, e-mail: fadak@dvo.ru,
Д. И. Харитонов, канд. техн. наук, стар. науч. сотр., ИАПУ ДВО РАН, г. Владивосток

Метод автоматической генерации скриптов в СУБД*

Рассматривается задача автоматического формирования скриптов и файлов на основе данных и шаблонов, хранящихся в базе данных. Эта задача является типичной для информационных систем, использующих в качестве средства хранения данных SQL-серверы. Авторами предложено оригинальное представление шаблонов файлов и структуры файлов для реляционных баз данных и метод генерации скриптов на основе этих представлений. Также описаны два примера генерации скриптов.

Ключевые слова: *автоматическая генерация кода программ, синтаксически управляемая трансляция, системы управления базами данных, предметно-ориентированные языки программирования, шаблонизатор, трансформация моделей, алгебра множеств*

Введение

В современном компьютерном мире все большую роль играют сложные информационные системы, такие как почтовые серверы, файлообменные серверы, сети банк-клиент, интернет-магазины, сети технической поддержки, социальные сети, энциклопедии и базы знаний. Эти системы уже сейчас предоставляют сервис, являющийся неотъемлемой частью повседневной жизни. Новые облачные технологии обещают стать неотъемлемой частью бизнеса с такими сервисами, как удаленный офис, сети мониторинга технических систем, распределенные системы доступа к вычислительным системам и хранилищам данных.

Общую идею функционирования информационных систем можно представить в виде трех перемежающихся в произвольном порядке шагов, состоящих из получения новых данных, обработки накопленных данных и формирования нового временного или постоянного контента, а также "отгрузки" данных конечным пользователям. По мере развития информационных систем "жесткие" программные структуры, отвечающие за интерфейсы и обработку данных, сменяются шаблонными механизмами и автоматически генерируемыми скриптами, позволяющими адаптировать системы ко все большему разнообразию внешних условий. Причем чем более интеллектуальный сервис реализуется в информационной системе, тем меньше жестких программных структур остается в ней. Таким образом, развитие сложных информационных систем неизбежно сталкивается с автоматической генерацией файлов или скриптов, и современные комплексные средства разработки web-сайтов, как правило, интегрируют в себе те или иные возможности автоматической генерации страниц и скриптов. В качестве примера можно отметить такие инструментальные средства организации сайтов, как MediaWiki, DokuWiki, MoinMoin, а также такие широкоизвестные платформы разработки сайтов (*web application framework*), как Zend Framework, Symfony, Yii.

Первоначальную идею автоматической генерации файлов представляет синтаксически управляемая трансляция [1], суть которой заключается в преобразовании исходного текста на основе правил разбора этого текста. С синтаксически управляемой трансляцией знакомят всех студентов факультетов прикладной математики и программирования. Однако с точки зрения разработки системы генерации метод синтаксически управляемой трансляции требует еще более высокого уровня знаний от программиста. Поэтому для автоматической генерации файлов были разработаны методы генерации на основе алгебры процессов [2], переписывания [3], трансформации моделей [4]. В теоретическом плане процесс генерации файлов развивается в направлении использования проблемно-ориентированных языков для описания модели или шаблона файла и дальнейшего преобразования этого файла с использованием компилятора, интерпретатора или другого механизма трансформации. По данным Википедии

* Работа выполнена при финансовой поддержке грантов №№ 12-I-P14-03, 12-I-П18-03 президиума РАН.

сейчас насчитывается более восьми десятков [5] самостоятельных программных средств автоматической генерации файлов, таких как Cheetah Template Engine, StringTemplate, AutoGen, используемых на практике. Они относятся к классу так называемых шаблонизаторов (*template processors* и *template engine*), их активно используют для генерации исходного кода, веб-страниц на различных языках, писем для рассылок, а также других видов форматированного текста. При этом основой генерации файлов являются шаблоны, представляющие собой небольшие программы или скрипты на специфических для каждого шаблонизатора языках. Для достижения большей производительности многие шаблонизаторы выполняют на основе шаблонов компиляцию генератора текстов. К основным недостаткам шаблонизаторов можно отнести требование знания пользователем языка описания или программирования, на котором пишут шаблоны, а также специализацию шаблонизаторов на определенных видах файлов, таких как, например, web-страницы. Большое число шаблонизаторов свидетельствует о том, что техническое решение задачи генерации файлов имеет меньшее значение, чем удобство и простота ее использования в конкретных условиях.

Несмотря на значительное число средств автоматической генерации файлов, практически не представляется возможным найти средства, разработанные для применения внутри СУБД, т. е. такие, в которых и шаблон файла, и генератор файлов были бы реализованы средствами СУБД. Учитывая, что ответственность за структуризацию данных в информационных системах несут базы данных, задачу автоматического формирования файлов для таких систем целесообразно размещать как можно ближе к самим данным. За счет этого можно, во-первых, сократить общие издержки на пересылку данных к генератору файлов, а во-вторых, контролировать целостность системы в случае изменения структуры данных. В настоящей статье авторами приводится метод генерации скриптов, ориентированный на применение именно в рамках СУБД и неоднократно использованный в системе управления заданиями на кластере [6—8].

Изложение материала в статье построено следующим образом. В первом разделе формально представлен метод генерации скриптов. Во втором разделе описана реализация метода для реляционных баз данных, приведены спецификации таблиц и взаимосвязей между ними, описаны основные стадии процесса генерации скриптов. В третьем разделе приведены два примера генерации скриптов с подробным изложением распределения данных в СУБД. В заключении сделаны выводы по применимости метода, а также определены направления дальнейшего развития его программной реализации.

1. Методы формирования скриптов

Опишем метод формирования скрипта на основе алгебры множеств. Для этого введем следующие обозначения и определения. Множество всех конечных последовательностей, составленное из символов множества A, включая пустую строку ε, обозначим как A^*. Конкатенация двух строк $u, w \in A^*$ записывается как uw. Обозначим множество $\dot{A}$, дополненное символом ε, как $\dot{A}$. Множество всех подмножеств множества A обозначим как $\mathcal{P}(A)$. Везде, где не оговорено явно, будем считать функции, определенные на множестве A (например, f), определенными также и на множествах $\dot{A}$, $\mathcal{P}(A)$, A^*. Причем значением любой функции для элемента ε является ε. Значением функции от последовательности элементов является последовательность значений ($\forall \xi \in A^* \Rightarrow f(\xi) = \{x_i = f(a_i) | a_i \in \xi\}$). А значением функции от подмножества элементов является множество значений функции от элементов подмножества ($\forall \xi \in \mathcal{P}(A) \Rightarrow f(\xi) = \{f(a) | a \in \xi\}$).

Будем считать заданными следующие множества: $\boldsymbol{P}$ — множество параметров, $\boldsymbol{R} \equiv \{req, opt, list, choice\}$ — множество ролей, $\boldsymbol{D}$ — пространство значений параметров, а также некоторый алфавит $\boldsymbol{A}$.

Определение 1. *Скрипт и скриптовая функция.* Без ограничения общности определим понятие скриптовой функции $\phi \in \boldsymbol{D}^* \to A^*$ как функции от последовательности значений[1], а также понятие вызова скриптовых функций как набора $\langle \rho, \phi \rangle$, где $\rho \in (\boldsymbol{P} \cup A^*)$ — последовательность переменных, заданных либо значением, либо параметром, а ϕ — скриптовая функция. Пространство всех вызовов скриптовых функций обозначим как σ. Тогда скриптом является последовательность вызовов скриптовых функций $\vartheta \in \sigma$.

Определение 2. *Шаблонный граф параметров скриптового файла.* Шаблонный граф параметров G определяется как набор $\langle V_g, E_g, E'_g, g_0, f \rangle$, где:

- V_g — множество вершин графа с выделенной корневой вершиной $g_0 \in V_g$;
- $f: V_g \to \boldsymbol{P} \times \boldsymbol{D} \times \mathcal{P}(\boldsymbol{R}) \times \mathbb{N} \times \sigma \times \sigma$ — функция, определяющая переменную узла, значение переменной, роль переменной, приоритет и скрипты узла соответственно:

$$\forall g \in V_g \Rightarrow f(g) = \langle f_p(g), f_d(g), f_r(g), f_n(g), s_{\text{in}}(g), s_{\text{out}}(g) \rangle;$$

- $E_g \subseteq V_g \times V_g$ — множество направленных ребер графа, на котором определяется функция инцидентности $()^{\bullet}: V_g \to V_g^*$, так что $\forall g \in V_g : g^{\bullet} = \{g_i \mid i \in [1...m], m \in \mathbb{N}, (g, g_i) \in E_g$, причем $\forall j < k \in [1...m] \Rightarrow g_j \neq g_k, f_n(g_j) \leqslant f_n(g_k)\}$;
- $E'_g \subset E_g$ есть покрывающее дерево графа: $\forall g \in V_g$: $g \neq g_0 \Rightarrow \exists! \{g_i \in V_g \mid i \in [0...n], n \in \mathbb{N}, g_n = g, \forall i \in [1...n] \Rightarrow$ $\Rightarrow (g_{i-1}, g_i) \in E'_g$, причем $\forall (g, g') \in E_g, f_r(g') = req \Rightarrow$ $\Rightarrow (g, g') \in E'_g\}$.

Последнее условие в определении констатирует, что для любой вершины существует единственный

[1] Согласно методу, известному в λ-исчислениях как карринг, функцию от k-переменных можно рассматривать как функцию от последовательности из k значений.

выделенный путь до корневой вершины. Таким образом, множество вершин шаблонного графа параметров файла образует дерево, дополненное "лишними" связями.

Определение 3. *Дерево параметров (структура) скриптового файла.* Дерево параметров файла T определяется как набор $T = \langle G_g, V_t, E_t, v_0, t\rangle$, где:

- $G_g = \langle V_g, E_g, E'_g, v_g, f\rangle$ — шаблонный граф параметров файла; v_g — вершина шаблонного графа;
- V_t — множество вершин дерева с корневой вершиной $v_0 \in V_t$;
- $t: V_t \to V_g \times \boldsymbol{D} \times \mathbb{N}$ — функция, определяющая прототип узла в шаблоне, значение переменной и приоритет узла соответственно: $\forall v \in V_t \Rightarrow t(v) = \langle \tau(v), t_d(v), t_n(v)\rangle$;
- $E_t \subseteq V_t \times V_t$ — множество направленных дуг дерева, причем:
  - для всех вершин определена функция инцидентности $()^{\cdot}: V_t \to V_t^*$, так что $\forall v \in V_t: v^{\cdot} = \{v_i \mid i \in [1...m], m \in \mathbb{N}, (v, v_i) \in E_t$, причем $\forall j < k \in [1...m] \Rightarrow v_j \neq v_k, (f_n(v_j) < f_n(v_k)) \vee (f_n(v_j) = f_n(v_k)) \wedge (t_n(v_j) \leqslant t_n(v_k))\}$;
  - для любой вершины существует единственный путь до корневой вершины: $\forall v \in V_t, v \neq v_0 \Rightarrow \exists! \{v_i \in V_t | v_n = v, 0 < i \leqslant n \in \mathbb{N} \Rightarrow (v_i, v_{i-1}) \in E_t\} \equiv rt(v)$;
  - каждому ребру дерева параметров ставится в соответствие одно ребро в шаблонном графе параметров: $\forall (v_1, v_2) \in E_t \Rightarrow (\tau(v_1), \tau(v_2)) \in E_g$.

Расширим область определения функций f на V_t:

$$\forall v \in V_t \Rightarrow f(\tau(v)) = \langle f_p(\tau(v)), f_d(\tau(v)), f_r(\tau(v)), f_n(\tau(v)), s_{\text{in}}(\tau(v)), s_{\text{out}}(\tau(v))\rangle.$$

Определим для узлов дерева функцию $S_p: V_t \times V_g \to V_t^*$, возвращающую последовательность узлов, связанных с данным и имеющих заданный прототип:

$$S_p(v, v_g) \equiv \{v_i \in V_t | i \in [0...n], n \in \mathbb{N}, (v, v_i) \in E_t, \tau(v_i) = v_g, \forall i > 0 \Rightarrow f_n(v_i) \geqslant f_n(v_{i-1})\}.$$

Определим также для узлов дерева функцию $S_r: V_t \times \boldsymbol{R} \to \mathcal{P}(V_t)$, возвращающую множество узлов, связанных с данным и имеющих заданную роль:

$$S_r(v, r) \equiv \{v' \in V_t | (v, v') \in E_t, r \in f_r(v')\}.$$

Тогда дерево параметров файла будем считать корректным, если:

$$\forall v \in V_t, v_g \in \tau(v)^{\cdot}, req \in f_r(v_g): \|S_p(v, v_g)\| \geqslant 1;$$

$$\forall v \in V_t, v_g \in \tau(v)^{\cdot}, lst \notin f_r(v_g): \|S_p(v, v_g)\| \leqslant 1;$$

$$\forall v \in V_t, v_g \in \tau(v)^{\cdot}, choice \in f_r(v_g): \|S_r(v, choise)\| = 1.$$

Таким образом, в корректном дереве все узлы имеют свои обязательные (*req*) параметры. Каждый узел имеет не более одного значения опциональных (*opt*) параметров, только один из альтернативных (*choice*) параметров и любое количество списковых (*lst*) параметров.

Определение 4. *Алгоритм генерации скрипта.* Будем считать заданным дерево параметров $T = \langle G, V_t, E_t, v_0, t\rangle$ и шаблонный граф параметров $G = \langle V_g, E_g, E'_g, g_0, f\rangle$.

Определим следующие вспомогательные функции для узлов дерева параметров:

- $\chi_p: V_t \times \boldsymbol{P} \to \dot{V}_t: \chi_p(v, p) = \begin{cases} v, f_p(v) = p \\ v', v' \in v^{\cdot} : f_p(v') = p \wedge f_r(v') \neq lst \\ \varepsilon, \text{ в противном случае;} \end{cases}$
- $\chi: V_t \times \boldsymbol{P} \to \dot{V}_t$, возвращающую узел дерева параметров для определенного параметра:

$$\chi(v, q) = \begin{cases} v', v' \in rt(v) \wedge \chi_p(v') \neq \\ \neq \varepsilon \wedge (\nexists v'' \in rt(v): v' \in rt(v'') \wedge \chi_p(v'') \neq \varepsilon \\ \varepsilon, \text{ в противном случае;} \end{cases}$$

- $D_p: V_t \times (\boldsymbol{P} \cup A^*) \to (\boldsymbol{P} \cup A^*)$ — преобразование параметра скриптовой функции:

$$D_p(v, q) = \begin{cases} q, q \in A^* \vee \chi(v, q) = \varepsilon \\ t_d(\chi(v, q)), \chi(v) \neq \varepsilon; \end{cases}$$

- $D_s: \sigma^* \times V_t \to \sigma^*$ — преобразование скрипта:

$$\forall \vartheta = \{\langle \rho_i, \phi_i\rangle\} \in \sigma^* \Rightarrow D_s(\vartheta, v) = \{\langle D_p(\rho_i, v), \phi_i\rangle\}.$$

Алгоритм генерации скрипта определяется через рекуррентную функцию Φ:

$$\forall v \in V_t \Rightarrow \Phi(v) = D_s(s_{\text{in}}(v))\Phi(v^{\cdot})D_s(s_{\text{out}}(v)).$$

Тогда конечный скрипт для заданных дерева параметров T и шаблона параметров G будет определяться как $\Phi(\langle v_0\rangle)$.

Предложенный в настоящем разделе метод генерации скриптов использует в качестве входной информации описание структуры скриптового файла в виде дерева параметров. Правила построения дерева параметров и шаблон генерации скрипта объединены в шаблонном графе параметров. Между структурой скриптового файла и шаблонным графом существует отображение, ставящее в соответствие каждому узлу и каждому ребру структуры один узел и одно ребро шаблонного графа. Алгоритм построения скрипта состоит в обходе дерева параметров скриптового файла в глубину (*depth-first search*) и добавлении в конечный скрипт преобразованных вызовов скриптовых функций из шаблонного графа на входе и выходе из каждого узла.

2. Реализация

Рассмотрим реализацию методов формирования скриптов в СУБД. На рис. 1 представлены схемы отношений основных таблиц базы данных. Среди таблиц можно выделить условно постоянные, которые модифицирует только администратор при модернизации всей системы. Они расположены в правой части данного рисунка и их имя начинается со слова SCRIPT. Таблица SCRIPT_TYPE содержит типы скриптов, поддерживаемых системой. Тип скрипта определяет следующие его характеристики.

- Назначение скрипта. В зависимости от назначения используется определенный набор скриптовых функций, полный список которых содержит таблица SCRIPT_FUNC. Таблица SCRIPT_FPARAMS хранит информацию о том, являются ли параметры функции входными или выходными.
- Вид конечного файла скрипта. Для различных типов скриптов предусматривают различные процедуры подготовки к генерации и завершения генерации.
- Множество входных параметров для скрипта хранится в таблице SCRIPT_TPARAMS. Во время процедуры подготовки к генерации этим параметрам будут присвоены значения.

Таблица SCRIPT_STAGE хранит признаки разделения скриптов на входной и выходной (s_{in}, s_{out} из разд. 1).

Шаблонный граф параметров скриптового файла описывается шестью таблицами: FTEMPLATE, FTEMPLATE_GRAPH, FTEMPLATE_GLINK, NODE_ROLE, TEMPLATE_SCRIPT и CALL_PARAM. Данные таблицы изменяют только администраторы системы для модификации существующих или добавления новых шаблонов, метаинформация о которых хранится в таблице FTEMPLATE. При любых изменениях таблиц шаблонного графа параметров изменяется поле CHANGE_TIME в таблице FTEMPLATE, которое в паре с полем COMPILE_TIME отвечает за актуальность шаблонного графа. Значение поля COMPILE_TIME указывает момент, когда было завершено редактирование шаблонного графа. В таблице FTEMPLATE_GRAPH хранится множество узлов графа (V_g из разд. 1), а в таблице FTEMPLATE_GLINK — множество ребер графа (E_g из разд. 1). Специальным флагом в данной таблице помечены те ребра, которые являются частью покрывающего дерева графа (E'_g из разд. 1). С помощью таблицы NODE_ROLE каждому узлу задается его роль, причем у одного узла может быть несколько ролей (***R*** из разд. 1). Параметры, используемые вызовами функций данного шаблона в качестве выходных, хранятся в таблице FTEMPLATE_PARAMS, которая заполняется процедурой завершения редактирования шаблонного графа. В таблице TEMPLATE_SCRIPT каждому узлу шаблонного графа ставится в соответствие множество последовательностей вызовов скриптовых функций. Разным типам скриптов из таблицы SCRIPT_TYPE и разным стадиям, описанным в SCRIPT_STAGE, соответствуют разные последовательности вызовов. Параметры вызова функций хранятся

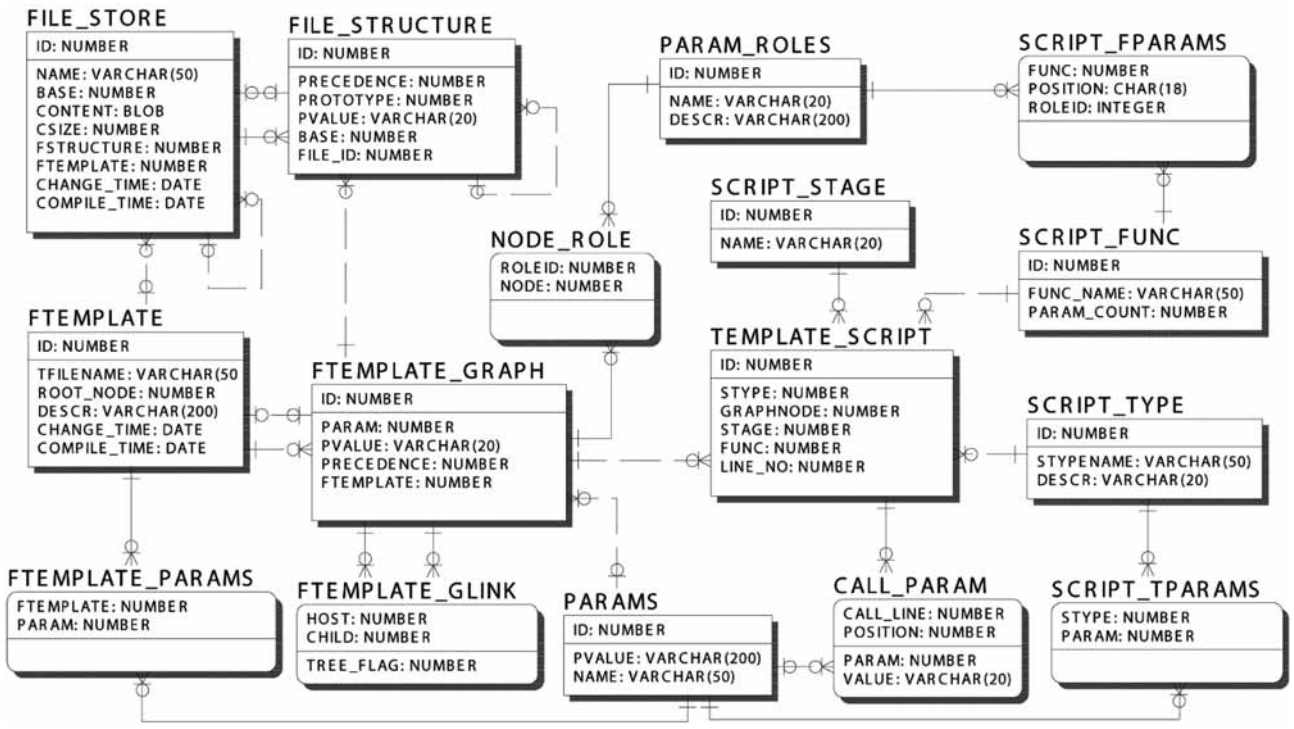


Рис. 1. Диаграмма отношений таблиц на SQL-сервере

в таблице CALL_PARAM либо в поле PARAM, либо в поле VALUE. Если значение поля PARAM не определено, то в качестве параметра вызова используется значение из поля VALUE, иначе используется поле PARAM, ссылающееся на список параметров в таблице PARAMS. Система генерации скриптов была реализована для двух различных версий SQL-сервера: ORACLE personal edition и PostgreSQL server. Прагматика использования системы генерации показывает, что скриптовые функции для скриптов различного назначения необходимо группировать. В частности, для ORACLE SQL-серверов скриптовые функции удобно группировать в пакеты, в PostgreSQL server можно использовать одинаковый префикс для функций одной группы или изолировать функции в разные схемы.

Таблица FILE_STRUCTURE используется для описания дерева параметров скриптового файла. При заполнении данной таблицы системой проверяются ограничения корректности, описанные в разд. 1. Обязательные (*req* из разд. 1) параметры узла заводятся автоматически. Запрещено иметь больше одного экземпляра параметра, если тип параметра отличен от *lst* (из разд. 1). Запрещено использовать более одного параметра типа *choice* (из разд. 1) у родительского узла. На основании дерева параметров генерируется скрипт, который записывается в таблицу FILE_STORE. Для выполнения генерации значение поля COMPILE_TIME таблицы FTEMPLATE должно превосходить значение поля CHANGE_TIME этой же таблицы. Результат выполнения полученного скрипта также будет сохраняться в таблицу FILE_STORE.

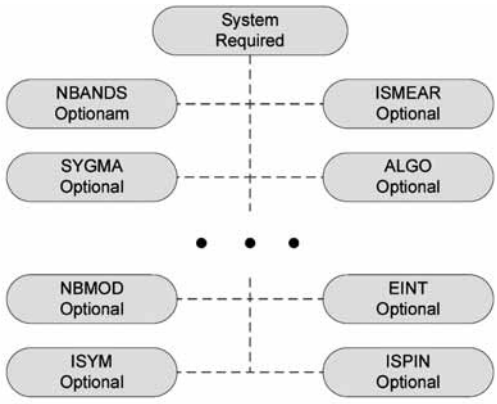
3. Простой пример генерации скрипта

В настоящем примере демонстрируется возможность использования системы генерация скриптов в целях построения файла настроек для расчетной программы. В качестве примера взят INCAR-файл настроек для программы квантово-механических расчетов в молекулярной динамике. Этот файл является основным входным файлом для пакета VASP, определяющим, "что делать, и как делать". Он записывается в простом теговом формате: каждая строка начинается с тега (строки), продолжается знаком "=" и далее одним или несколькими значениями. Большинство параметров вычислений имеют значения по умолчанию и могут не указываться.

```
System = Si(111) 1x1 - Cu

NBANDS = 576
ISMEAR = 1
SIGMA  = 0.1
ALGO   = All
NSW    = 100
ISIF   = 2
IBRION = 2
LREAL  = A
NELM   = 60
ENCUT  = 300
LPARD = True
NBMOD = -3
EINT = 0.4
ISYM = 0
ISPIN = 2
```

а)



б)

Рис. 2. Примеры INCAR-файла (*а*) и варианта шаблонного графа (*б*)

ID	PRECEDENCE	PROTOTYPE	BASE	PVALUE
131	0	System		Si(111) 1x1 – Cu
132	1	NBANDS	131	576
133	2	ISMEAR	131	1
134	3	SIGMA	131	0.1
135	4	ALGO	131	All
136	5	NSW	131	100
137	6	ISIF	131	2
138	7	IBRION	131	2
139	8	LREAL	131	A
140	9	NELM	131	60
141	10	ENCUT	131	300
142	11	LPARD	131	True
143	12	NBMOD	131	-3
144	13	EINT	131	0.4
145	14	ISYM	131	0
146	15	ISPIN	131	2

Рис. 3. Представление дерева параметров в табличной форме

Предположим, что необходимо получить конечный файл, представленный на рис. 2, *а*. В первую очередь, необходимо построить шаблонный граф параметров скриптового файла. Такой граф для генерации INCAR-файлов будет выглядеть так, как представлено на рис. 2, *б*. Все узлы шаблонного графа, кроме корневого, имеют опциональный характер. Каждый узел, кроме корневого, соответствует одной строке конечного файла. Название параметра узла соответствует наименованию тега соответствующей строки. Для каждого из узлов (тега) можно определить значение по умолчанию.

Следующим шагом на основании шаблонного графа параметров скриптового файла необходимо построить дерево параметров скриптового файла. Для построения этого дерева нужен пользовательский интерфейс, который может быть реализован либо с использованием web-технологий, либо в виде специальной клиентской программы. Однако построение пользовательского интерфейса не является частью настоящей статьи. Со стороны SQL-сервера система генерации скриптов контролирует условия корректности получаемого дерева параметров, описанные в разд. 1. Результирующее дерево параметров целиком содержится в таблице FILE_STRUCTURE, на рис. 3 представлено множество записей из этой таблицы, соответствующих скрипту генерации конечного файла на рис. 2.

После получения дерева параметров скриптового файла необходимо выполнить процедуру генерации скрипта. В основе этой процедуры находятся скрипты s_{in} и s_{out}, связанные с каждым из узлов шаблонного графа параметров. Вследствие относительной про-

```
declare
fileID number:=172;
TheLob clob;
begin
SBS.FileID:=FileID;
---------------------------------
TXT.AddLine('System',' = ',
'Si(111) 1x1 - Cu');
TXT.AddLine;
TXT.AddLine('NBANDS',' = ', '576');
TXT.AddLine('ISMEAR',' = ', '1');
TXT.AddLine('SIGMA' ,' = ', '0.1');
TXT.AddLine('ALGO'  ,' = ', 'All');
TXT.AddLine('NS'    ,' = ', '100');
TXT.AddLine('ISIF'  ,' = ', '2');
TXT.AddLine('IBRION',' = ', '2');
TXT.AddLine('LREAL' ,' = ', 'A');
TXT.AddLine('NELM'  ,' = ', '60');
TXT.AddLine('ENCUT' ,' = ', '300');
TXT.AddLine('LPARD' ,' = ', 'True');
TXT.AddLine('NBMOD' ,' = ', '-3');
TXT.AddLine('EINT'  ,' = ', '0.4');
TXT.AddLine('ISYM'  ,' = ', '0');
TXT.AddLine('ISPIN' ,' = ', '2');
---------------------------------
update FILE_STORE f set
f.content=TheLob where f.id=fileId;
dbms_lob.freetemporary(TheLob);
end;
```

а)

```
do $$declare
fileID integer:=172;
TheLob text='';
begin
---------------------------------------
TXT_AddLine(TheLob,'System',' = ',
'Si(111) 1x1 - Cu');
TXT_AddLine(TheLob);
TXT_AddLine(TheLob,'NBANDS',' = ', '576');
TXT_AddLine(TheLob,'ISMEAR',' = ', '1');
TXT_AddLine(TheLob,'SIGMA' ,' = ', '0.1');
TXT_AddLine(TheLob,'ALGO'  ,' = ', 'All');
TXT_AddLine(TheLob,'NS'    ,' = ', '100');
TXT_AddLine(TheLob,'ISIF'  ,' = ', '2');
TXT_AddLine(TheLob,'IBRION',' = ', '2');
TXT_AddLine(TheLob,'LREAL' ,' = ', 'A');
TXT_AddLine(TheLob,'NELM'  ,' = ', '60');
TXT_AddLine(TheLob,'ENCUT' ,' = ', '300');
TXT_AddLine(TheLob,'LPARD' ,' = ', 'True');
TXT_AddLine(TheLob,'NBMOD' ,' = ', '-3');
TXT_AddLine(TheLob,'EINT'  ,' = ', '0.4');
TXT_AddLine(TheLob,'ISYM'  ,' = ', '0');
TXT_AddLine(TheLob,'ISPIN' ,' = ', '2');
---------------------------------------
update FILE_STORE set content=TheLob,
csize=char_length(TheLob) where id=fileId;
end$$;
```

б)

Рис. 4. Скрипты генерации INCAR-файла:
а — для Postgress; *б* — для ORACLE

стоты структуры INCAR-файла, узлам шаблонного графа требуется только скрипт s_{in}.

Для формирования INCAR-файла реализована группа скриптов TXT. Ввиду технологических различий скрипты шаблонного графа параметров для разных SQL-серверов отличаются друг от друга. На рис. 4 приведены скрипты, произведенные системой генерации для двух SQL-серверов. Начало скриптов до строки комментария из символов "–" формируется на этапе подготовки к генерации скрипта и является жестко запрограммированным. Окончание скрипта после такого же комментария генерируется на этапе завершения формирования скрипта.

Приведенный в настоящем разделе пример описывает ситуацию, когда на выходе у системы генерации скриптов получается скрипт, предназначенный для формирования текстового файла настроек. Этот пример показателен тем, что стадии процесса генерации, а именно подготовка шаблонного графа скриптового файла и формирование дерева параметров скриптового файла, выглядят относительно легко автоматизируемыми.

4. Сложный пример генерации скрипта

Рассмотрим пример генерации скрипта в целях построения запроса на языке SQL для некоторой системы учета рабочего времени. Такие системы служат для построения отчетов о пребывании сотрудников на рабочем месте за определенный период времени по конкретному сотруднику, группе сотрудников или по сотрудникам, имена которых заданы определенным шаблоном, либо по подразделению или списку подразделений. При этом данные в отчете могут группироваться по дням недели, по неделям, по месяцам или за весь период. Для успешного построения отчета обязательно должен быть задан только период отчета, остальные ограничения и фильтры указывают опционально. Общее число различных SQL-запросов выборки данных для построения отчетов о пребывании сотрудников составляет более 40 вариантов.

В отличие от предыдущего примера для построения конечного запроса будет использовано два шаблона генерации скриптов. Такой подход позволяет разбить задачу на две независимых подзадачи. Первая подзадача контролирует ввод исходных данных пользователем, вторая — корректный синтаксис результирующего SQL-запроса. На рис. 5 представлены все

Рис. 5. Стадии генерации конечного скрипта на языке SQL

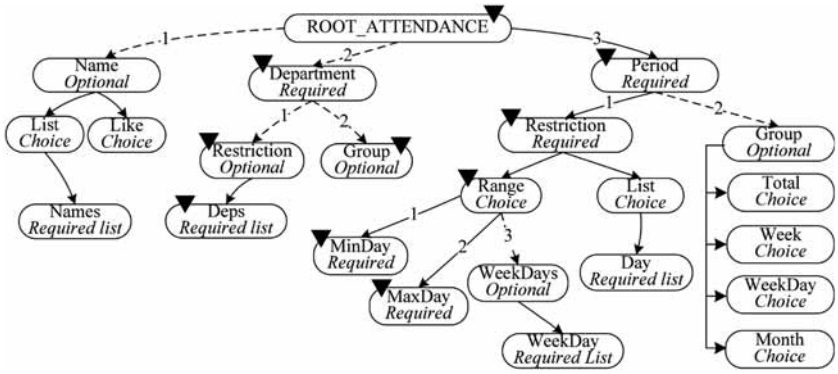


Рис. 6. Шаблонный граф параметров логики построения отчета учета времени

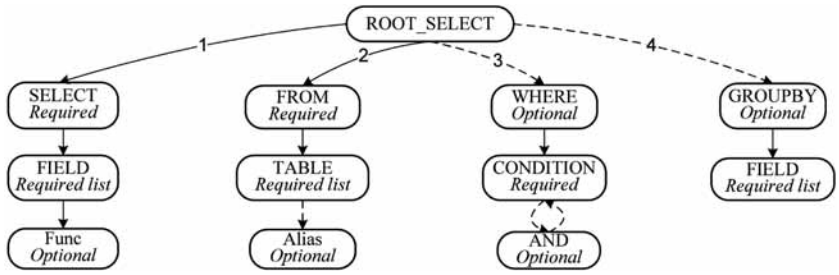


Рис. 7. Шаблонный граф параметров модели простого запроса SELECT

стадии построения запроса. На первой стадии администратор системы создает шаблоны генерации скриптов. Первый шаблон, отвечающий за схему построения пользовательского интерфейса, отображен на рисунке в области "Логика". Второй шаблон является моделью простого запроса SELECT в СУБД в области "Модель". На второй стадии пользователь вносит значения необходимых параметров для построения запроса в дерево параметров. Далее автоматически генерируется скрипт для построения дерева параметров модели запроса (третья стадия на рис. 5). При этом используются скриптовые функции из группы PFS, предназначенные для формирования дерева параметров по заданному шаблону. Четвертая стадия, указанная на рис. 5, заключается в выполнении подготовленного PFS-скрипта. Далее опять выполняется автоматическая генерация скрипта для построения запроса SELECT (пятая стадия), в результате выполнения которого получается конечный запрос.

На рис. 6 изображен шаблонный граф параметров логики построения отчета для учета рабочего времени. Сплошные стрелки на рисунке обозначают связи с обязательными узлами, а штриховые — связи с опциональными узлами. На стрелках указан приоритет узлов, используемый при обходе дерева параметров. Если приоритет узла не важен, например, для узлов с ролью *choice*, то он пропущен. Ниже названия узла курсивом приводится его роль. Логика построения отчета состоит в том, что пользователю необходимо, во-первых, обязательно указать либо период времени, либо список дней, для которых строится отчет; во-вторых, пользователь может указать фильтры по именам, по отделам, по дням недели, если указан период времени; в-третьих, пользователь может выбрать правила группировки по отделам и по периодам.

На рис. 7 представлен шаблонный граф параметров, соответствующий синтаксису простого запроса SELECT на языке SQL. Этот запрос может состоять только из разделов SELECT, FROM, WHERE и GROUPBY. В качестве логического оператора для связи условий используется только оператор AND. Для отображения графа использовались те же правила, что и для графа на рис. 6.

Рассмотрим процесс генерации конечного запроса для отчета для следующей гипотетической ситуации. Предположим, что пользователю необходимо построить сгруппированный по отделам (Department) отчет посещаемости для отделов под номерами 11, 21, 102 и 105 за период времени с 22.02.2013 по 22.04.2013. На рис. 6 черным треугольником помечены вершины, для которых пользователем должны быть определены значения в дереве параметров для рассматриваемой ситуации. На рис. 8 представлен вариант PFS-файла для СУБД ORACLE personal edition. На этапе подготовки к генерации PFS-файла автоматически заполняется область объявления локальных для этого скрипта переменных (строки 1—6 на рис. 8). При заходе в корневую вершину графа (вершина ROOT_ATTENDANCE на рис. 6) автоматически, независимо от введенных пользователем параметров, добавляются строки 7—16 на рис. 8, соответствующие входному скрипту для этой вершины. Далее по порядку обходятся вершины Restriction и список вершин Deps, располагающиеся за узлом Department. В результате этого обхода в PFS-файл добавляются строки 17—26. При обходе узла Group, также располагающегося за узлом Department, в PFS-файл добавляются строки 27—35. Эта часть скрипта удалит из списка полей запроса SELECT поле с ФИО сотрудника (e.Name) и добавит агрегатные функции SUM для полей ewt:RealTime, ewt:NeedTime, ewt:Delta. Далее осуществляется обход вершин, связанных с временным периодом, за который строится отчет: Period, Restriction, Range, MinDay и MaxDay. В PFS-файл добавляются строки 36—39 и строки 40—43, определяющие минимальные и максимальные условия для периода запроса (поля ewt:DatePeriod, соответственно). Окончательно, по возвращении в корневую вершину, добавляется 44-я строка, соответствующая выходному скрипту этой вершины. Процедура завершения генерации добавляет последнюю, 45-ю, строку в файл.

В результате выполнения PFS-файла строится дерево параметров модели простого запроса SELECT, которое мало отличается от шаблонного графа, представленного на рис. 7. В этом дереве используются все узлы шаблонного графа параметров модели, причем списковые узлы по несколько раз, а узлы CONDITION и AND

```
declare
  TheLob              clob;
  LCid                varchar2(50);
  Node                varchar2(50);
  Condition           varchar2(250);
begin
  PFS.BeginFile('SIMPLE_SELECT');
  PFS.AddValue('.SELECT.FIELD', 'e.Name');
  PFS.AddValue('.SELECT.FIELD', 'e.Department');
  PFS.AddValue('.SELECT.FIELD', 'ewt.RealTime');
  PFS.AddValue('.SELECT.FIELD', 'ewt.NeedTime');
  PFS.AddValue('.SELECT.FIELD', 'ewt.Delta');
  PFS.AddValue('.SELECT.FIELD', 'ewt.DatePeriod');
  PFS.AddValue('.FROM.TABLE',   'Employee e');
  PFS.AddValue('.FROM.TABLE',   'Employee_Working_Time ewt');
  PFS.AddValue('.WHERE.CONDITION', 'ewt.EmployeeID = e.EmployeeID', LCid);
  PFS.AddValue2(LCid, 'and');
  PFS.ClearStr(Condition);
  PFS.Add2Str(Condition, 'e.Department in (');
  PFS.Add2Str(Condition, '11,');
  PFS.Add2Str(Condition, '21,');
  PFS.Add2Str(Condition, '101,');
  PFS.Add2Str(Condition, '105,');
  PFS.TrimStr(Condition, ',');
  PFS.Add2Str(Condition, ')');
  PFS.AddValue2(LCid, 'CONDITION', Condition);
  PFS.DelValue('.SELECT.FIELD', 'e.Name');
  PFS.AddValue('.GROUPBY.FIELD', 'e.Department');
  PFS.Get('.SELECT.FIELD', 'ewt.RealTime', Node);
  PFS.AddValue2(Node, 'Func', 'sum');
  PFS.Get('.SELECT.FIELD', 'ewt.NeedTime', Node);
  PFS.AddValue2(Node, 'Func', 'sum');
  PFS.Get('.SELECT.FIELD', 'ewt.Delta', Node);
  PFS.AddValue2(Node, 'Func', 'sum');
  PFS.AddValue('.GROUPBY.FIELD', 'ewt.DatePeriod');
  PFS.AddValue2(LCid, 'and');
  PFS.ClearStr(Condition);
  PFS.Add2Str(Condition, 'ewt.DatePeriod > ', '22.02.2013');
  PFS.AddValue2(LCid, 'CONDITION',     Condition);
  PFS.AddValue2(LCid, 'and');
  PFS.ClearStr(Condition);
  PFS.Add2Str(Condition, 'ewt.DatePeriod < ', '22.04.2013');
  PFS.AddValue2(LCid, 'CONDITION', Condition);
  PFS.EndFile(TheLob);
end;
```

Рис. 8. Пример скриптового PFS-файла

```
select e.Department, sum(ewt.RealTime), sum(ewt.NeedTime), sum(ewt.Delta),
      sum(ewt.DatePeriod)
from Employee e, Employee_Working_Time ewt
where ewt.EmployeeID = e.EmployeeID and e.Department in (11, 21, 101, 105)
      and ewt.date > 22.02.2013 and ewt.date < 22.04.2013
group by e.Department, ewt.DatePeriod
```

Рис. 9. Готовый запрос на языке SQL

формируют цепочку. На основе шаблона и дерева параметров модели простого запроса SELECT процедура генерации скрипта подготавливает скриптовый PSQL-файл. Его структура подобна скриптам, изображенным на рис. 4. И наконец, после выполнения скриптового PSQL-файла, получается искомый запрос на языке SQL, который представлен на рис. 9.

Заключение

В настоящей статье представлен метод генерации скриптов с использованием СУБД на всех стадиях его формирования: при подготовке шаблона, при описании структуры скриптового файла и во время генерирования. Как правило, при автоматической генерации файлов используют шаблоны, записанные на проблемно-ориентированных языках в виде файлов. В статье предложен вариант описания шаблонов скриптов в табличной форме. При этом реляционные связи между таблицами позволяют описывать логику и контролировать различные характеристики конечного файла. Следует отметить, что для узлов шаблонного графа скриптового файла используется множество ролей, достаточное для описания синтаксических диаграмм Вирта. Это обстоятельство позволяет говорить об универсальности способа описания шаблонов для широкого спектра задач. В настоящий момент описанный метод генерации скриптов используется в системе управления вычислительной средой WBS (*Web batch system*), разрабатываемой авторами. Расширение областей применения метода генерации скриптов позволяет выделить эту подсистему WBS в качестве самостоятельного продукта, потенциально применимого в информационных системах на базе SQL серверов. Для формирования этого продукта авторами планируется, во-первых, развивать пользовательский интерфейс построения шаблонов, а во-вторых, автоматизировать построение.

Список литературы

1. **Ахо А. В., Лам М. С., Сети Р., Ульман Д. Д.** Компиляторы: принципы, технологии, инструменты. М.: Вильямс. 2008. 1184 с.

2. **Yingxu Wang, Xinming Tan, Cyprian F. Ngolah.** Design and Implementation of an Autonomic Code Generator Based on RTPA // International Journal of Software Science and Computational Intelligence. 2010. Vol. 2, № 2. P. 44—65.

3. **Bellegarde F.** Program Transformation and Rewriting // Rewriting Techniques and Applications, 4th International Conference, RTA-91. 10—12 April, 1991. Como, Italy. 1991. Vol. 488. P. 226—239.

4. **Hemel Z., Kats L. C. L., Groenewegen D. M., Visser E.** Code Generation by Model Transformation. A Case Study in Transformation Modularity // Software and Systems Modeling. 2010. Vol. 9, Is. 3. P. 375—402.

5. **Template engine.** Wikipedia. URL: http: // en.wikipedia.org/wiki/Template_engine

6. **Одякова Д. С., Тарасов Г. В., Харитонов Д. И.** Система WBS как расширенный инструмент управления вычислительной средой. Высокопроизводительные параллельные вычисления на кластерных системах // Материалы XII Всероссийской конференции "Высокопроизводительные параллельные вычисления на кластерных системах". 20—28 ноября 2012 г., г. Нижний Новгород / Под. ред. проф. В. П. Гергеля. Нижний Новгород: Изд-во Нижегородского госуниверситета, 2012. С. 300—304.

7. **Голенков Е. А., Левин В. А., Харитонов Д. И., Шиян Д. С.** Организация поддержки циклических расчетов в GRID-сегменте ДВО РАН // Горный информационно-аналитический бюллетень. 2009. Т. 18, № 12. С. 225—229.

8. **Бабяк П. В., Тарасов Г. В.** Организация потоковой обработки спутниковых данных на основе GRID-технологий // Горный информационно-аналитический бюллетень. 2009. Т. 18, № 12. С. 170—175.

УДК 004.4

В. В. Костюк, канд. техн. наук, проф., e-mail: kvv@rguitp.ru,
А. А. Овчинников, студент, Российский государственный университет инновационных технологий и предпринимательства, г. Москва

Опыт разработки сервис-ориентированных BIRT-отчетов на основе IBM-продуктов

Представлен опыт, полученный авторами при разработке приложения, позволяющего пользователям получать необходимые им выходные документы в форме отчетов на экран компьютера и на бумажный носитель. Разработанное приложение позволяет формировать и выдавать информационные документы средствами генератора отчетов BIRT, встроенного в среду разработки приложений компании IBM, а именно в Rational Software Architect.

Авторы надеются, что этот опыт будет полезен не только для студентов высших учебных заведений, но и для специалистов — приверженцев концепции сервис-ориентированной архитектуры и разработчиков подобных решений в разных предметных областях.

Ключевые слова: *программирование, технологии программирования, интеграция IBM-продуктов, конфигурирование вычислительной среды, генератор отчетов*

Как правило, автоматизированная информационная система включает такие приложения, как средства навигации по ее функциональным возможностям (меню), экранные формы ввода информации, пользовательские интерфейсы, информационные документы и отчеты, как результаты выполненных запросов. Такие результаты обычно представляются пользователям на экран или в бумажном виде [1].

В качестве примера разработки последнего из перечисленных выше сервис-ориентированных приложений взят фрагмент автоматизированной информационной системы управления движением пригородных электропоездов. Этот фрагмент предназначен для решения задачи представления расписания движения пригородных поездов Курского направления центрального региона России.

При построении системы использованы следующие программные продукты:

- IBM DB2 — реляционная СУБД;
- IBM Rational Data Architect (IBM RDA) — среда проектирования баз данных;
- IBM Rational Software Architect (IBM RSA) — интегрированная среда моделирования и разработки (версия 7.5.5);
- IBM WebSphere Application Server (IBM WAS) — сервер приложений;
- Eclipse BIRT — среда разработки и исполнения отчетов.

Разработка базы данных

Для решения отмеченной выше задачи представления расписания движения поездов необходимо было разработать базу данных. База данных расписания проектировалась под СУБД DB2. Схема базы данных TIMETABL в усеченном виде в качестве примера приведена на рис. 1.

Далее рассмотрим операции формирования этой базы данных. В качестве инструментария для разработки упомянутой базы данных был взят продукт IBM RDA. Разработка осуществлялась прямым проектированием, начиная с логического уровня, путем реализации составленных и заполненных вручную анкет [2]. Фрагмент анкетного описания сущностей (Entity), атрибутов и связей приведен в Приложении.

Следуя реляционной теории, в модели на логическом уровне связь между отношениями STATION (станции) и TRAIN (поезда) была представлена как множественная (*M*:*M*). Такая же множественная связь

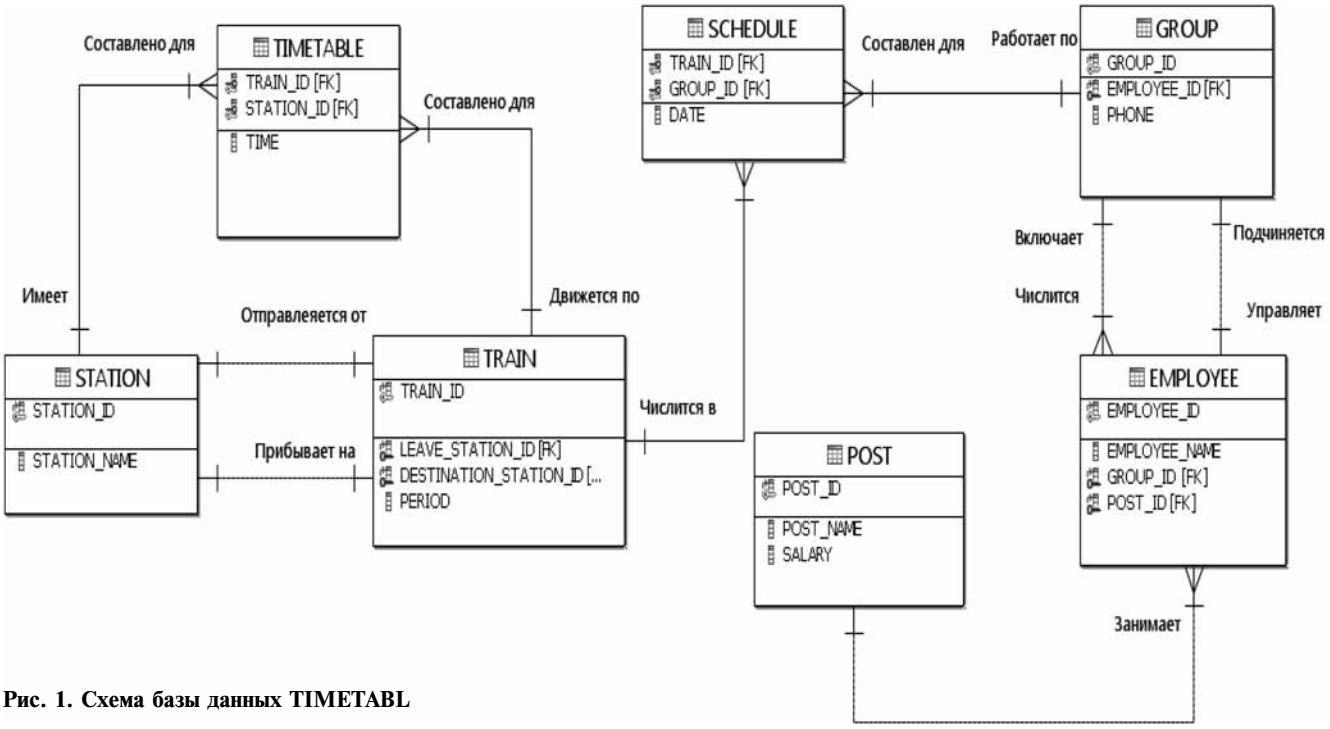


Рис. 1. Схема базы данных TIMETABL

представлялась между отношениями TRAIN (поезда) и GROUP (бригады).

На основе логической модели проводилась генерация физической модели с трансформацией множественных связей в идентифицирующие связи к новым ассоциативным сущностям, которые создавались в результате генерации физической модели базы данных. Также в результате генерации в физической модели имена сущностей и атрибутов заменялись на их англоязычные синонимы в соответствии с наличием в анкетном описании установки "Persistent: v" и "Abbreviation: Group".

После генерации физической модели необходимо было новые сгенерированные сущности (TIMETABLE, SCHEDULE) дополнить новыми атрибутами (TIME, DATE) и описать их. На основании полученной таким образом физической модели в среде IBM RDA осуществлялась генерация DDL-файла этой физической модели. Далее на основании дальнейших преобразований DDL-файла была создана (сгенерирована) схема базы данных TIMETABL (рис. 1) в среде СУБД DB2 для последующего заполнения ее данными, необходимыми для решения задач, которыми (в качестве примера в рамках рассматриваемой предметной области) являлись следующие:

- формирование расписания движения электропоездов для конкретной станции;
- формирование расписания движения конкретного электропоезда;
- формирование отчетных документов (список станций, список поездов, расписание движения поездов).

Разработка web-сервиса

На начальной стадии реализации рассматриваемого примера разработки сервис-ориентированного приложения выполнялись шаги по схеме, которая описывает содержание лабораторного практикума "Разработка программного обеспечения информационных систем на основе интеграции IBM-продуктов" по курсу "Информационные технологии" [6]. В этом лабораторном практикуме приведенные ниже шаги разработки выполнялись в среде инструментария IDE Eclipse. В данной статье кратко рассмотрены вопросы выполнения этих шагов в среде IBM RSA [4]. Осуществлялась следующая последовательность шагов, каждый из которых требовал создания своего проекта (рис. 2):

- разработка JPA-сущностей (JPA — Java Persistence API) для работы с базой данных (проект TimetableJPA);
- разработка компонентов Enterprise Java Beans (EJB) бизнес-логики (проект TimetableEJB);
- разработка web-интерфейса приложения для тестирования бизнес-логики (проект TimetableWeb);
- разработка web-сервиса (проект TimetableWebService);
- разработка клиентского сервис-ориентированного приложения для тестирования web-сервиса (проект TimetableClient).

Для реализации шагов, связанных с формированием сервис-ориентированных BIRT-отчетов, требовалось создание таких проектов, как TimetableWebService, TimetableReport (рис. 2).

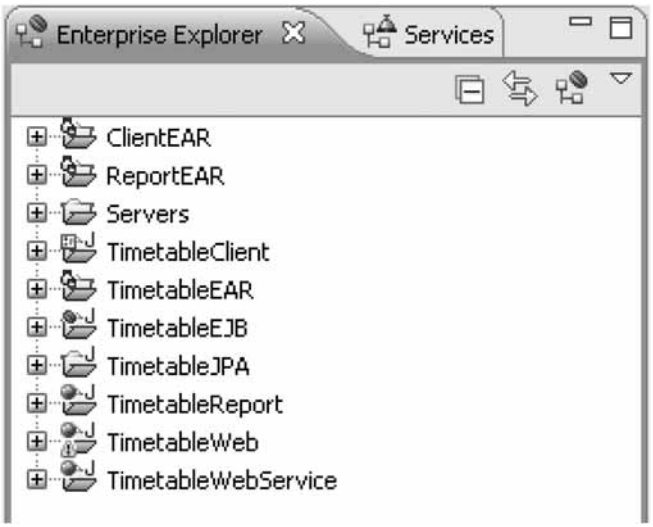


Рис. 2. Состав проектов, реализующих задачу рассматриваемой предметной области

Для работы приложения с базой данных используется технология JPA, которая связывает таблицы и данные с классами и объектами Java. Связывание (ORM — object relation mapping) осуществляется путем добавления специальных аннотаций к коду классов. Такие классы, также называемые JPA-сущностями, генерируются средствами IBM RSA.

Бизнес-логика реализуется как набор EJB-компонентов — специальным образом аннотированных Java-классов, реализующих обработку данных. Эти компоненты поддерживают распределенное выполнение операций и использование механизма транзакций для обеспечения целостности данных.

Разработанное корпоративное приложение, представляющее компонент сервиса, размещается на сервере приложений IBM WAS. Размещение и тестирование приложения проводится средствами IBM RSA.

Для тестирования EJB-контейнера было написано простое web-приложение в проекте TimetableWeb (рис. 2) с использованием технологии JSP [6].

В качестве унифицированного интерфейса взаимодействия с компонентами бизнес-логики используется web-сервис, который был реализован в проекте TimetableWebService (рис. 2). Данный web-сервис разработан методом "сверху вниз", т. е. сначала создавалось описание на WSDL (Web Service Definition Language), а по нему средствами IBM RSA генерировался каркас кода сервиса. Файл WSDL составляет более четырех страниц, поэтому в данной статье не приводится.

Метод "сверху вниз" выбран по причине того, что в этом случае web-сервис выступает не только в качестве интерфейса для вызова методов обработки, но и как слой, где происходит преобразование данных для передачи клиентскому приложению.

Использование web-сервиса в качестве источника данных для генератора отчетов Eclipse BIRT

Продукт BIRT (Business Intelligence and Reporting Tools) [7] предназначен для создания, редактирования и выдачи отчетов и является открытым проектом EclipseFoundation [3]. Этот продукт доступен в различных вариантах, в частности, как плагин к основанным на Eclipse продуктам (в том числе и IBM RSA). В качестве источника данных для BIRT могут выступать различные объекты и системы, например XML-файл, база данных, EJB-контейнер или web-сервис. Это эффективное инструментальное средство для создания отчетов в корпоративных сетях, удобное для интеграции с другими приложениями.

Подсистема формирования BIRT-отчетов по движению пригородных электропоездов использует web-сервис как источник данных и включает:

- web-приложение, содержащее формы отчетов;
- среду выполнения отчетов — BIRT Engine.

Данная подсистема размещается на сервере IBM WAS либо на любом контейнере сервлетов, например Apache Tomcat.

Доступ к описанной системе осуществляется посредством универсального клиента — web-браузера. С его помощью можно просматривать на экране отчеты, а также экспортировать их в файлы широко распространенных форматов (например, DOC, XLS, ODT, PDF) или отправлять на печатающее устройство. При этом web-сервис скрывает особенности реализации системы и выступает в роли API, позволяя работать с данными другим системам или клиентским приложениям.

Выше было рассмотрено проектирование базы данных в среде IBM RDA и создание ее в среде СУБД DB2. В среде RSA в контейнере EAR собраны такие элементы, как JPA, EJB, web-сервис и web-приложение в качестве теста EJB-элемента. Отдельно в той же среде RSA средствами Eclipse BIRT сформированы сервис-ориентированные отчеты и проверена их локальная визуализация. Структура представляемой в данной статье системы изображена на рис. 3.

Для того чтобы теперь сформированные сервис-ориентированные BIRT-отчеты были доступны как

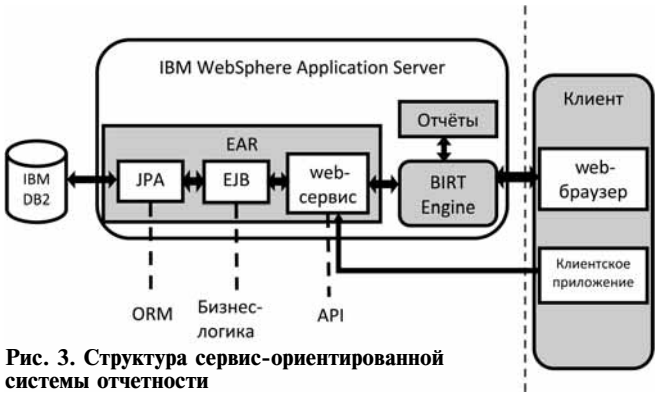


Рис. 3. Структура сервис-ориентированной системы отчетности

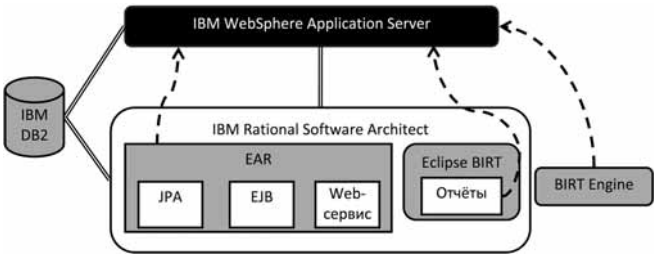


Рис. 4. Инфраструктура интегрированной среды исполнения BIRT-отчетов

Рис. 5. Пример начальной страницы подсистемы отчетов

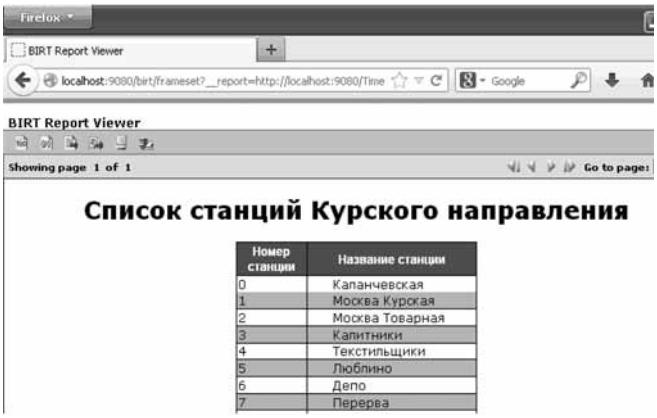


Рис. 6. Пример отчета "Список станций"

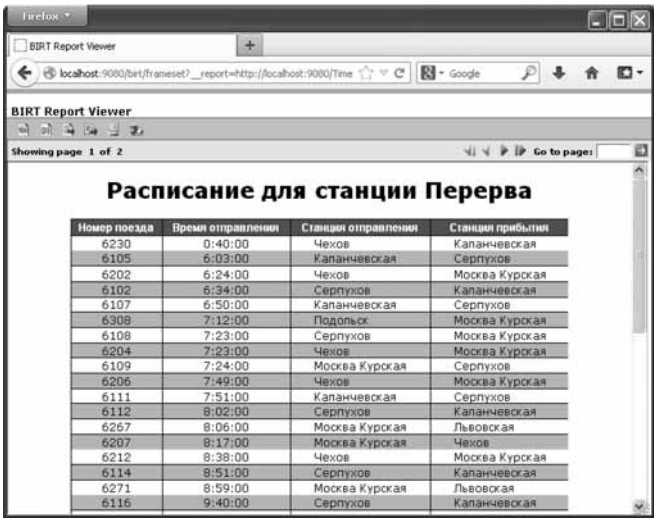


Рис. 7. Пример отчета "Расписание для станции"

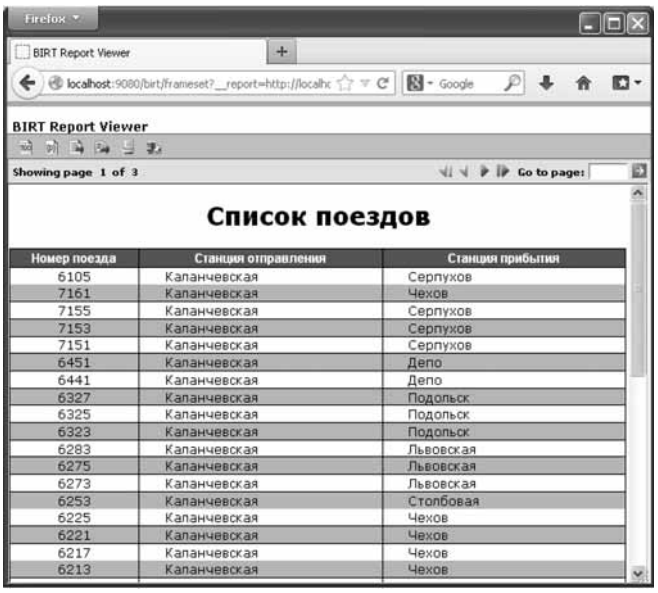


Рис. 8. Пример отчета "Список поездов"

сервис-ориентированное приложение клиент-серверного уровня, необходимо обеспечить размещение всех элементов приложения на сервер, как показано на рис. 4. Визуализация и выдача отчетов на печать в клиент-серверном решении осуществляются при исполнении отчетов средствами BIRT Engine, интегрированными в сервер WebSphere Application Server средствами административной консоли WAS.

Работа с готовой системой осуществляется через web-браузер (рис. 5).

На рис. 6—8 приведены примеры генерируемых системой отчетов.

Также поддерживается распределение компонентов системы (база данных, корпоративное приложение, подсистема отчетов) на разные компьютеры, что обеспечивает дополнительную гибкость.

Список литературы

1. **Современные** технологии создания программного обеспечения. Обзор. URL: http://citforum.ru/programming/application/program/lit.shtml.

2. **Биберштейн Н., Боуз С., Джонс К., Фиаммант., Ша Р.** Компас в мире сервис-ориентированной архитектуры (SOA): ценность для бизнеса, планирование и план развития предприятия. Пер. с англ. М.: КУДИЦ-ПРЕСС, 2007. 256 с.

3. **Введение** в интегрированную среду разработки Eclipse. URL: http://www.javaportal.ru/java/ide/intro_eclipse.html.

4. **Фанг Дж., Йу К., Лау К.** и др. Введение в IBM Rational Application Developer. Учебное руководство (в комплекте CD). Пер. с англ. М.: КУДИЦ-ПРЕСС, 2006. 592 с.

5. **Костюк В. В.** История с программированием. Его величество Код, вокруг да около него // Программная инженерия. 2011. № 2. С. 39—47.

6. **Костюк В. В., Ушанов М. А.** К организации лабораторного практикума "Разработка программного обеспечения информационных систем на основе интеграции IBM-продуктов" // Программная инженерия. 2012. № 5. С. 15—20.

7. **Business** Intelligence and Reporting Tools. URL: http://projects.eclipse.org/projects/birt.

Приложение

Фрагмент анкетного описания сущностей (Entity), атрибутов и связей базы данных

Сущность "Станция"

Name: Станция
Label: Станц
Persistent: v
Abbreviation: Station
Owner: is_x1_09
Documentation:

1. Определение: Один из остановочных пунктов на маршруте следования поезда
2. Дополнительное определение: Совокупность станций образует направление железной дороги
3. Примеры возможных запросов:
   a. Список всех станций данного направления
   b. Список поездов, пунктом прибытия которых является некоторая станция
   c. Сведения о времени отправления поезда
4. Примеры экземпляров сущности: Москва Курская, Текстильщики, Бутово
5. Идентификатор сущности: Номер станции.

Атрибуты сущности

1. ***Атрибут "Номер станции" сущности "Станция"***
   Name: Номер станции
   Label: Ном_стан
   Abbreviation: Station_id
   Data Type: Привязка к домену "Номер станции"
   Ключ: PK
   Режим нулевых значений: Not null.
2. ***Атрибут "Название станции" сущности "Станция"***
   Name: Название станции
   Label: Назв_стан
   Abbreviation: Station_name
   Data Type: VarChar Привязка к домену "Номер станции"
   Length/Presision: 25
   Режим нулевых значений: Not null.

Связи

1. ***Связь "Станция — Поезд"***
   Verb Phrase со стороны дочерней сущности — отправляется от
   Documentation: Поезд начинает свое движение от станции отправления
   Тип связи: неидентифицирующая обязательная
   Кардинальность связи 1.

2. ***Связь "Станция — Поезд"***
   Verb Phrase со стороны дочерней сущности — прибывает на
   Documentation: Поезд заканчивает свое движение на станции прибытия
   Тип связи: неидентифицирующая обязательная
   Кардинальность связи 1, ∞.

УДК 004.9, 659.1

Ю. И. Петров, канд. экон. наук, стар. препод., e-mail: mailtoyuripetrov@gmail.com,
Московский государственный университет приборостроения и информатики,
С. Н. Макаров, аналитик-экономист, ООО "Гифтилайф", г. Москва

Некоторые аспекты автоматизации сферы интернет-маркетинга на примере компании ООО "Гифтилайф"

Обозначены особенности интернет-маркетинга, рассмотрены отдельные вопросы деятельности организаций в данной сфере. На примере интернет-маркетинговой компании ООО "Гифтилайф" приведено описание его бизнес-процессов по работе с клиентами. Рассмотрены особенности автоматизации этих процессов и аспекты технической реализации разработанной в компании информационной системы.

Ключевые слова: *автоматизация, интернет-маркетинг, бизнес-процесс, информационная система, программное обеспечение, Delphi, Microsoft Access, стандарты IDEF0, IDEF1X.*

Введение

В современном мире высокий уровень конкуренции заставляет предприятия и организации искать новые пути повышения эффективности их бизнеса. В случае если рынком функционирования таких структур являются глобальные сети, основной и наиболее известной из которых является сеть Интернет, уровень конкуренции увеличивается на порядок. Кроме того, Интернет может быть использован для проведения анализа спроса и предложения, а также выбора услуг и товаров, что непосредственно затрагивает область интернет-маркетинга — комплекса мероприятий по продвижению и продаже на рынке товаров и услуг с помощью сетевых технологий [1, 2]. Инструментальные средства интернет-маркетинга значительно отличаются от традиционных средств такого назначения и имеют преимущества, заключающиеся в широчайшем охвате целевой аудитории, персонализации взаимодействия с клиентами и снижении транзакционных издержек.

В настоящее время интернет-маркетинг используется на предприятиях различной отраслевой направленности [3—5]. При этом зачастую предприятия предпочитают обращаться в организации, которые специализируются на оказании подобного рода услуг. Рассматриваемая в настоящей статье компания ООО "Гифтилайф" работает на рынке интернет-маркетинга с 2012 г. За короткий срок существования компания обрела стабильных партнеров в этом бизнесе и предоставляет клиентам широкий список услуг, в числе которых:

— маркетинговое сопровождение;
— продвижение сайта;
— выставочный консалтинг;
— брендинг;
— полиграфия;
— тренинги и мастер классы.

В настоящее время в ООО "Гифтилайф" весь объем входящей и исходящей (отчетной по проектам) информации обрабатывается сотрудниками вручную или с использованием стандартных программных продуктов Microsoft Word и Excel, которые не адаптированы под специфику работы организации. В связи с этим руководство компании поставило вопрос об использовании в организации информационной системы (ИС), которая бы позволила повысить эффективность работы компании.

1. Существующая технология работы с клиентами

Прежде всего следует обозначить специфику деятельности компании. Ключевым процессом является процесс работы с клиентами. Используя стандарт IDEF0, представим структуру и схемы бизнес-процессов, соответствующих работе менеджера данного процесса в компании ООО "Гифтилайф". Контекстная диаграмма процесса работы с клиентом в модели AS-IS ("как есть") представлена на рис. 1.

Рис. 1. Контекстная диаграмма "Работа с клиентом"

На рис. 2 представлена детализация данного бизнес-процесса.

Процесс работы с клиентами в ООО "Гифтилайф" включает:

- составление предварительной заявки на рекламный интернет-проект;
- маркетинговое исследование интернет-сферы заказчика;
- составление коммерческого предложения и заключение договора;
- реализацию рекламного интернет-проекта;
- составление отчетов.

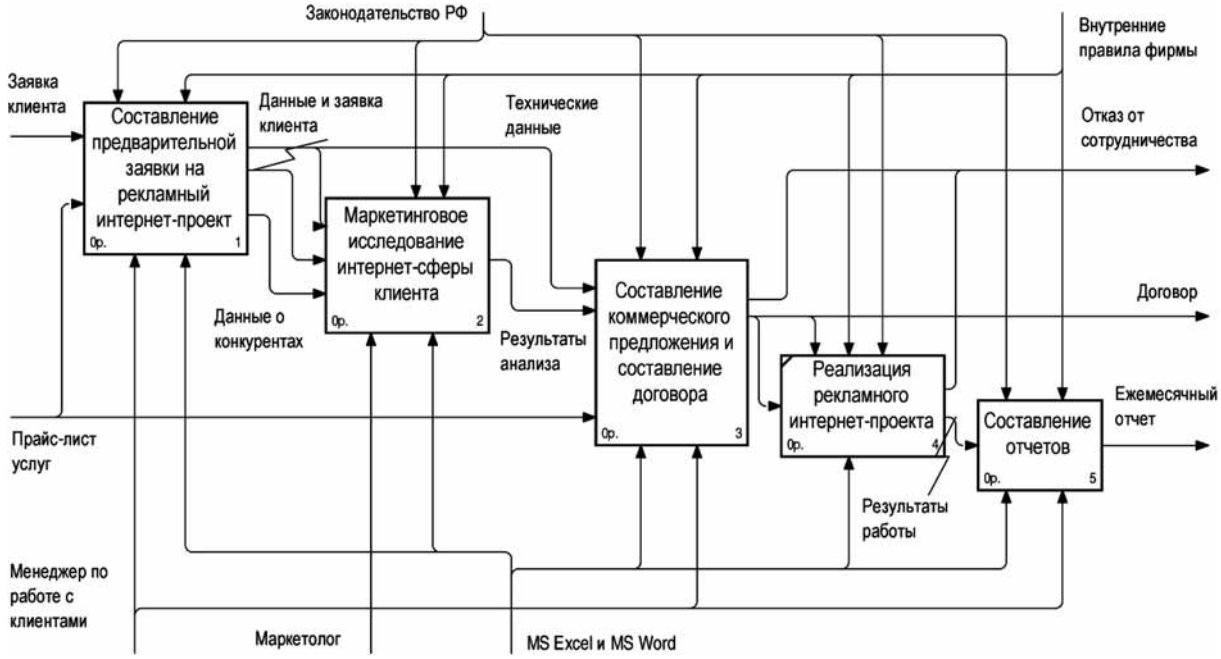


Рис. 2. Детализация процесса "Работа с клиентом"

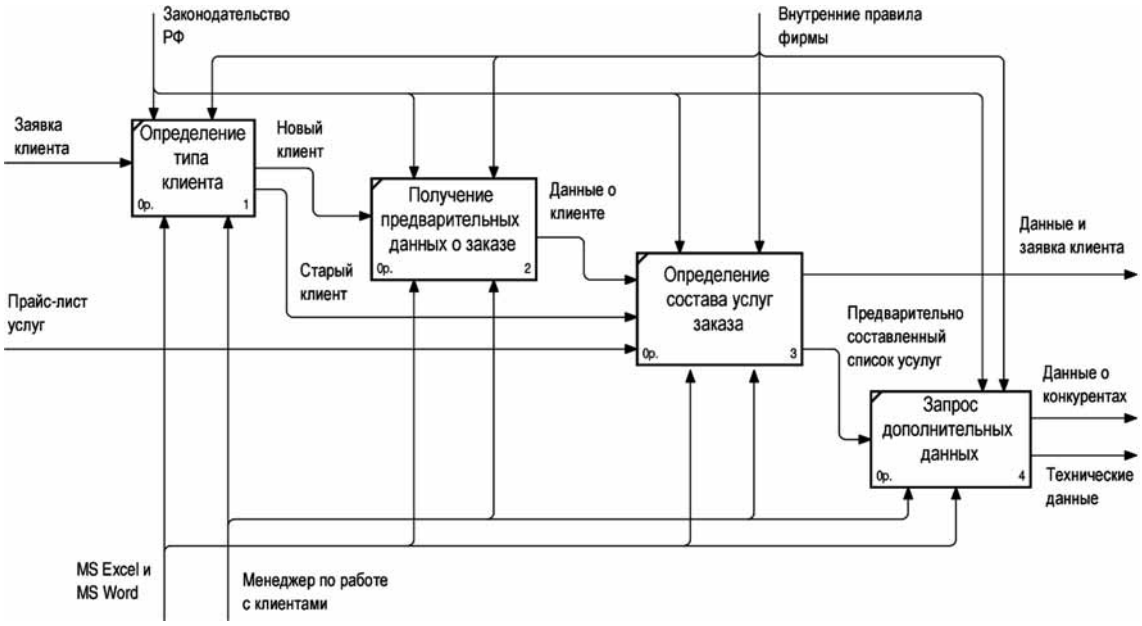


Рис. 3. Детализация процесса "Составление предварительной заявки на рекламный интернет-проект"

Во время работы с клиентом менеджер заполняет клиентскую базу, оформляет и следит за исполнением проекта, после чего составляет различные отчеты для клиента посредством стандартного пакета программ Microsoft.

На рис. 3 представлен процесс составления предварительной заявки на рекламный интернет-проект.

На данном этапе менеджер по работе с клиентами, в первую очередь, определяет тип клиента. Если клиент уже сотрудничает с ООО "Гифтилайф", менеджер сразу переходит к составлению списка услуг. Если клиент новый — вначале заполняют первичные данные, такие как ФИО клиента, сфера деятельности, контактный телефон и адрес электронной почты. В дальнейшем, после определения состава необходимых услуг, менеджер может запросить дополнительные данные, например:

- текущие показатели компании клиента;
- список основных конкурентов (если известен);
- реквизиты сайта;
- различные статистические данные, выгруженные из директа сайта.

Под директом подразумевают подключенные к сайту системы контекстной рекламы (в частности, Яндекс.Директ и Google Adwords), из которых может осуществляться выгрузка данных.

На рис. 4 представлена детализация процесса маркетингового исследования интернет-сферы заказчика.

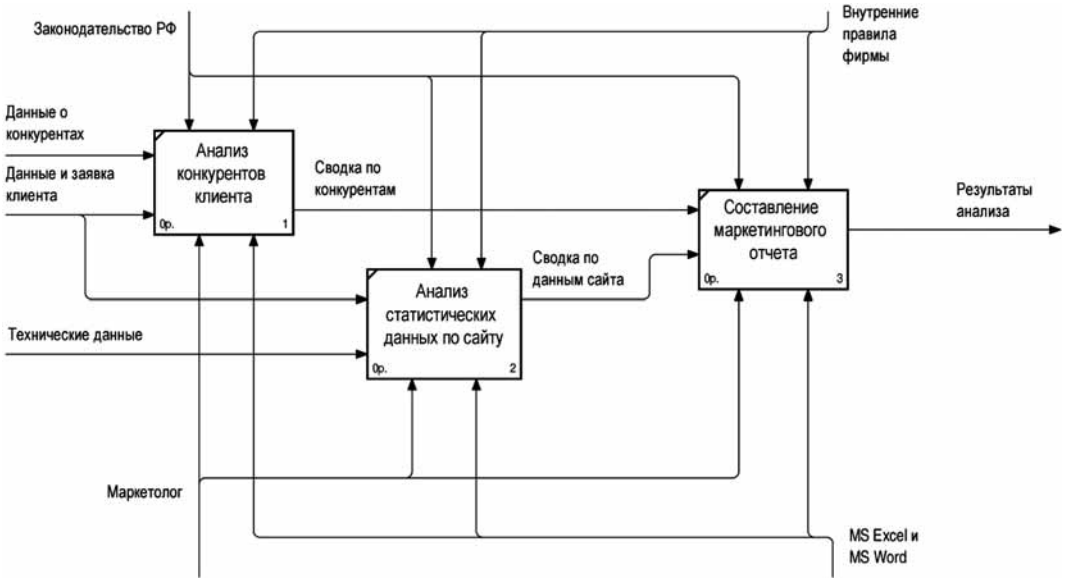


Рис. 4. Детализация процесса "Маркетинговое исследование интернет-сферы заказчика"

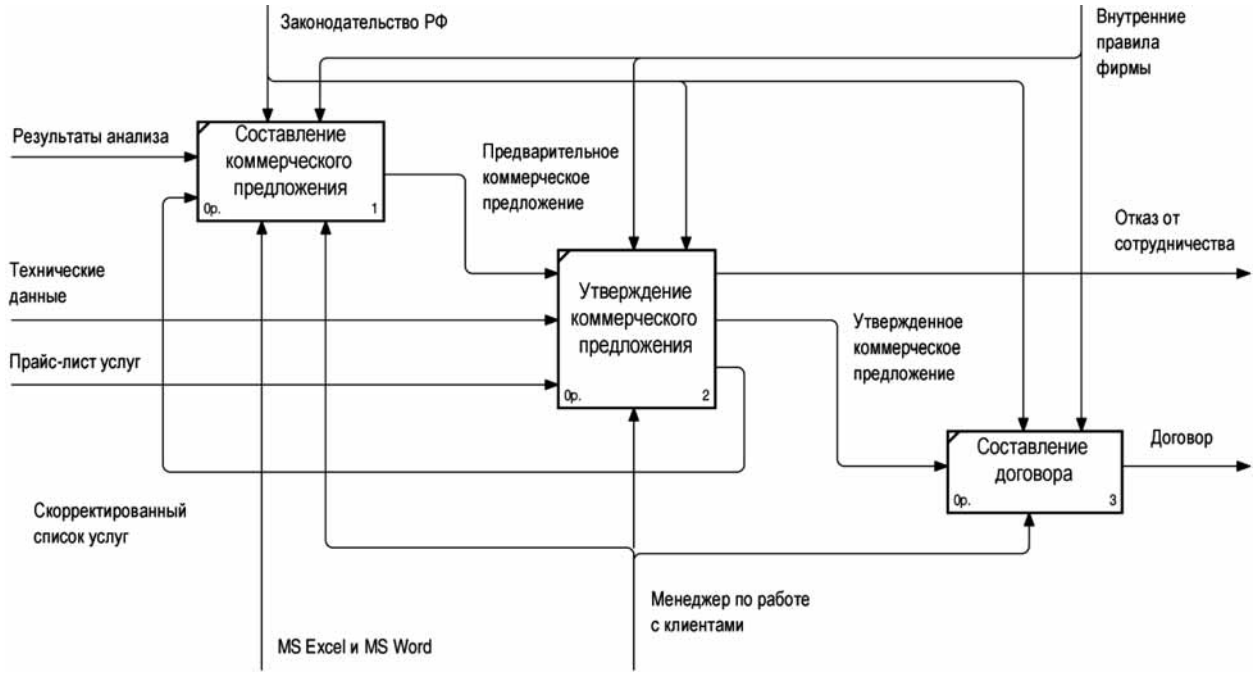


Рис. 5. Детализация процесса "Составление коммерческого предложения и составления договора"

На данном этапе маркетолог, опираясь на первичные данные, указанные в заявке, проводит анализ сферы деятельности заказчика. В первую очередь, выполняют анализ деятельности конкурентов, после специализируют статистические данные по сайту. Здесь оценивают такие показатели, как посещаемость сайта, прибыль, полученная за счет привлеченных клиентов на сайт, число внутренних и внешних ссылок, а также в целом оценивают структуру, дизайн и контент сайта. На заключительном этапе составляется маркетинговый отчет, в котором маркетолог описывает текущее состояние фирмы заказчика, указывает основные направления для развития, а также предлагает методы по улучшению фирмы заказчика. Данный отчет маркетолог передает менеджеру по работе с клиентами для определения окончательного списка услуг для клиента.

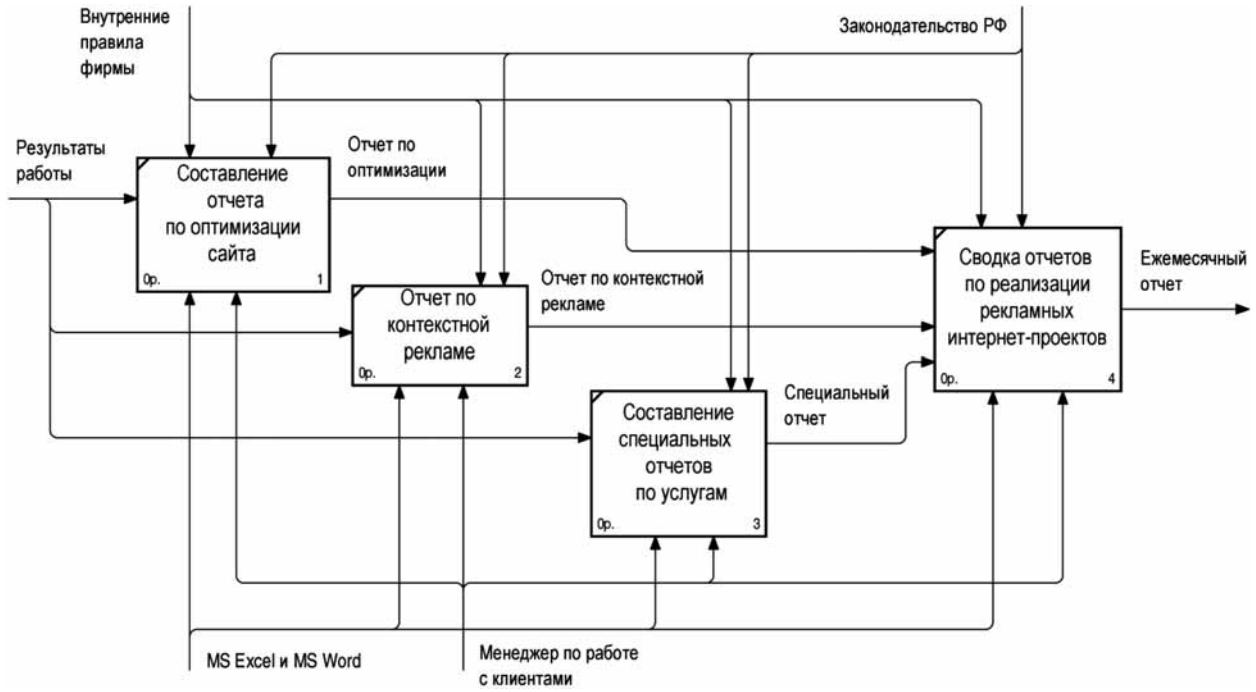


Рис. 6. Детализация процесса "Составление отчетов"

На рис. 5 представлена детализация процесса составления коммерческого предложения и составления договора.

На основании данных заявки и проведенного маркетингового анализа формируют итоговый список услуг для клиента, на основе которого составляют коммерческое предложение. Если клиента по каким-либо причинам не устраивает коммерческое предложение компании, оно идет на пересмотр и процесс повторяется. Следующим является этап составления и согласования всех пунктов договора, который происходит в тесном контакте с клиентом и основан на переговорах даже по самым мелким вопросам. Вначале составляют первичный вариант договора по обслуживанию клиента, а после финальных корректировок договор подписывают клиент и генеральный директор. Как только договор подписан, начинают проводить работы по сайту. По результатам обслуживания составляют отчеты, которые в дальнейшем предоставляют клиенту.

На рис. 6 представлена детализация процесса составления отчетов.

Составление отчетов о проведенной работе является одним из важнейших этапов в работе с клиентами. Компания предоставляет следующие отчеты:

- по оптимизации сайта;
- по директу сайта;
- по продажам клиента;
- по посещаемости сайта.

В отчетах также указывают все работы, которые проводились по проекту на протяжении определенного периода времени и представляют динамику улучшения показателей ресурсов клиента. Последнее важно для того, чтобы заказчик мог отследить результаты работы компании, свои затраты, а также мог убедиться в эффективности интернет-рекламы.

2. Требования к разрабатываемой информационной системе

Требующаяся агентству специализированная ИС должна являться мощным инструментом, который поддержит и улучшит работу менеджера по работе с клиентами. Создаваемая ИС должна представлять собой автоматизированную систему, которая позволит осуществлять гибкий поиск по базе данных и рассчитывать необходимые экономические показатели.

Основные требования, предъявляемые к системе:

- обеспечение централизованного накопления, хранения и обработки данных, необходимых для автоматизации бизнес-процессов, рассмотренных в разд. 1;
- наличие интуитивно понятного интерфейса — важное свойство, так как при возможном расширении штата сотрудников новые сотрудники должны тратить минимальное время на освоение системы.

Система также должна содержать базу данных клиентов, услуг, сотрудников компании и шаблоны документов (коммерческих предложений, договоров и отчетов). Разрабатываемая ИС должна иметь модуль отчетности, позволяющий выводить на экран и печать отчеты, обозначенные в разд. 1, а также договор на оказание услуг.

Пользователей ИС предлагается разделить на две категории: менеджеры, имеющие возможность полноценного ведения проектов, и администраторы, контролирующие доступ сотрудников к ИС.

3. Новая технология работы с клиентами

Как уже было отмечено ранее, в ООО "Гифтилайф" было принято решение о разработке и внедрении ИС, которая позволила бы значительно оптимизировать работу менеджера по работе с клиентами. Использование ИС в процессе работы агентства привело к необходимости реинжиниринга рассмотренных бизнес-процессов. На рис. 7 представлена контекстная диаграмма работы с клиентом в модели TO-BE ("как будет") с учетом использования ИС.

Детализация данного процесса представлена на рис. 8.

Ключевым моментом автоматизации бизнес-процессов является параллельное введение "личной карты" клиента, которую заполняют и корректируют начиная с момента принятия заказа и до составления окончательного варианта договора с клиентом. В качестве положительного побочного эффекта внедрения такой карты следует отметить повышение общей детерминированности процесса работы с клиентами. С помощью лич-

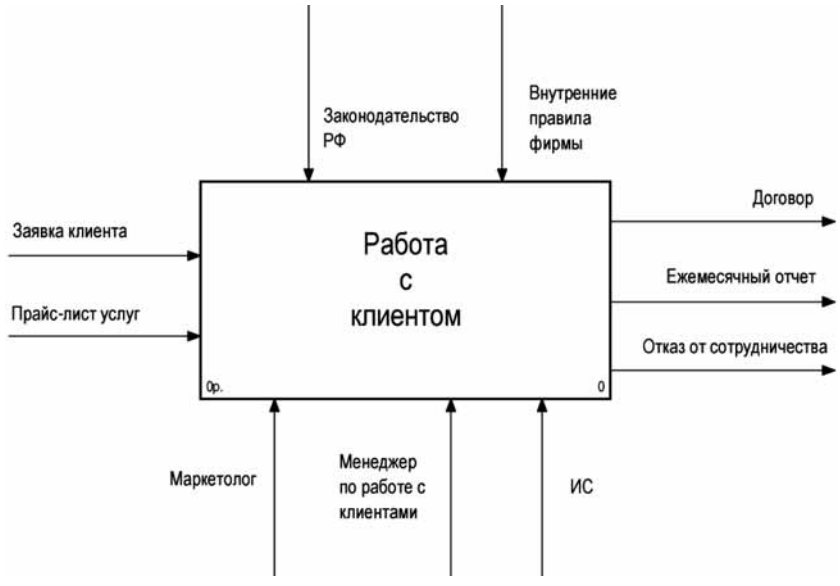


Рис. 7. Контекстная диаграмма "Работа с клиентом"

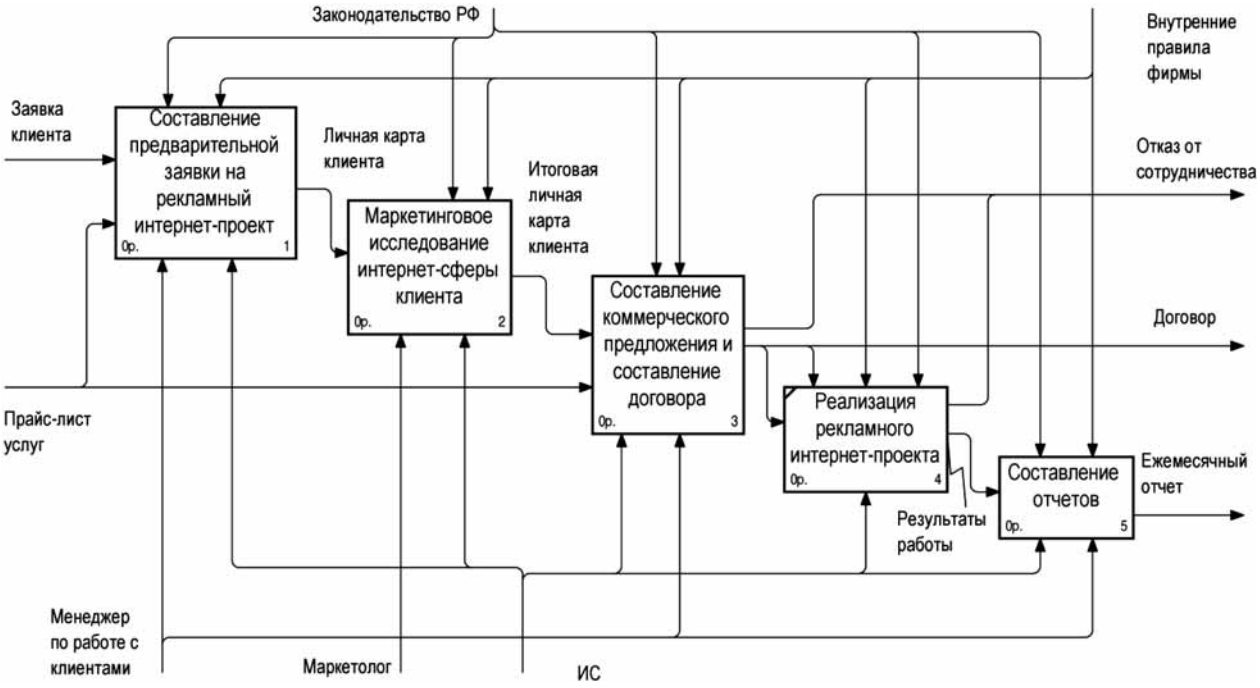


Рис. 8. Детализация процесса "Работа с клиентом"

ной карты клиента ИС подсчитывает окончательную стоимость услуг, предоставляемых компанией, а также формирует отчеты. Подробно принцип работы с личной картой клиента будет рассмотрен далее.

Детализация процесса составления предварительной заявки на рекламный интернет-проект представлена на рис. 9.

На первоначальном этапе менеджер по работе с клиентами по-прежнему должен выяснить потребности клиента. Благодаря ИС существует возможность быстро определить, обращался ли клиент ранее в компанию или нет. В зависимости от этого личную карту клиента либо корректируют, либо создают снова. На данном этапе подобная карта содержит персональные данные и список услуг, в которых нуждается клиент.

После заполнения личной карты начинается процесс анализа текущего состояния интернет-бизнеса заказчика.

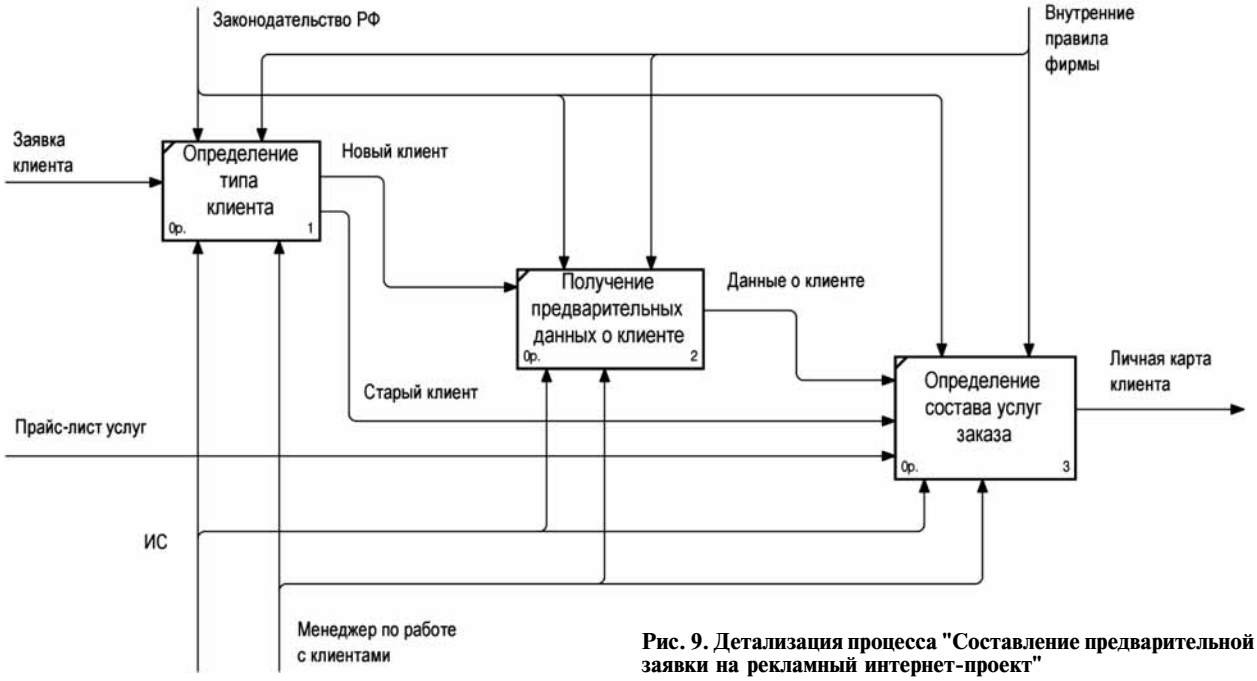


Рис. 9. Детализация процесса "Составление предварительной заявки на рекламный интернет-проект"

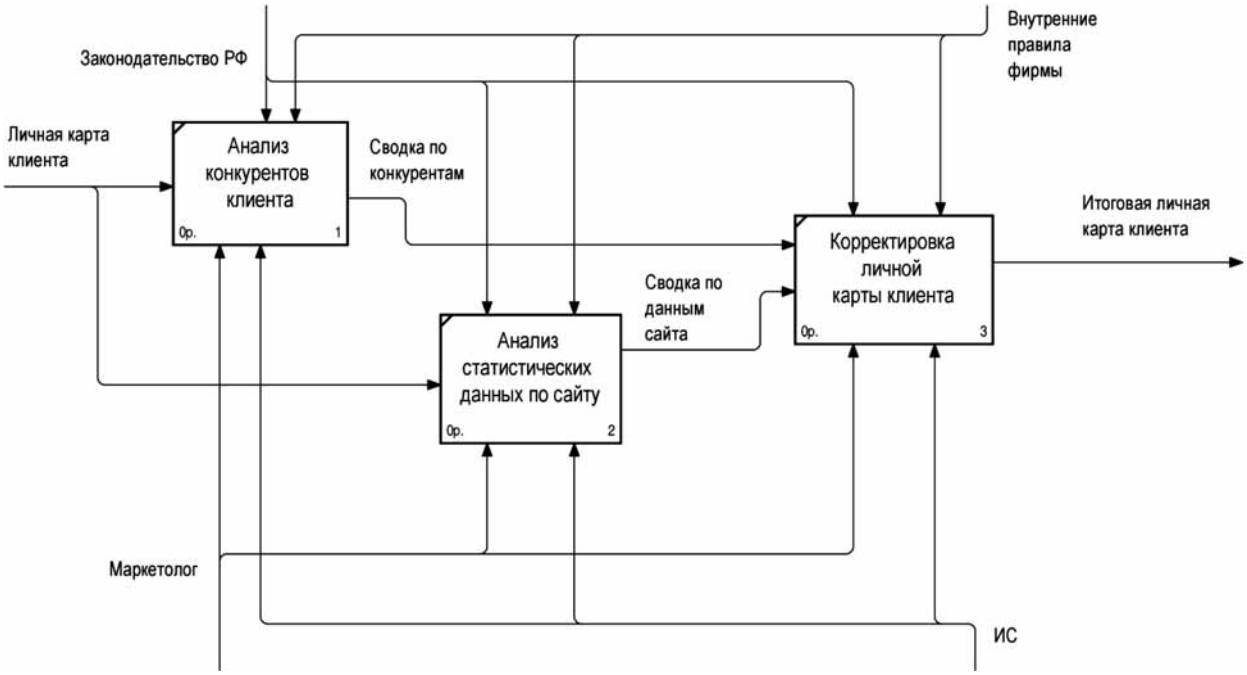


Рис. 10. Детализация процесса "Маркетинговое исследование интернет-сферы заказчика"

Детализация процесса маркетингового исследования интернет-сферы заказчика представлена на рис. 10.

Получив личную карту клиента, маркетолог начинает проводить анализ сферы деятельности заказчика (для услуг, указанных в заявке). По окончании анализа, с помощью ИС маркетолог заполняет анкету по текущему состоянию интернет-маркетинга у клиента. В анкете он указывает, какие услуги и в каком объеме необходимо предложить и оказать клиенту, а какие услуги не являются необходимыми.

После проведения анализа личную карту клиента передают менеджеру по работе с клиентами для составления коммерческого предложения. Детализация

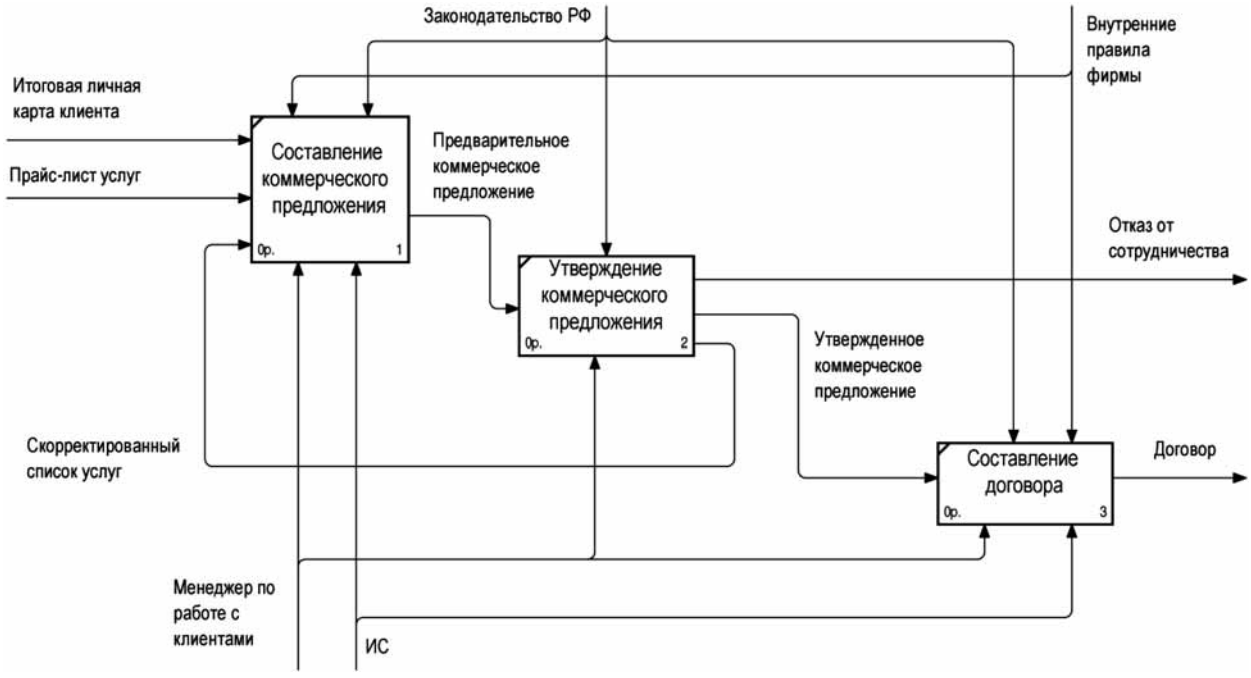


Рис. 11. Детализация процессов "Составление коммерческого предложения и составление договора"

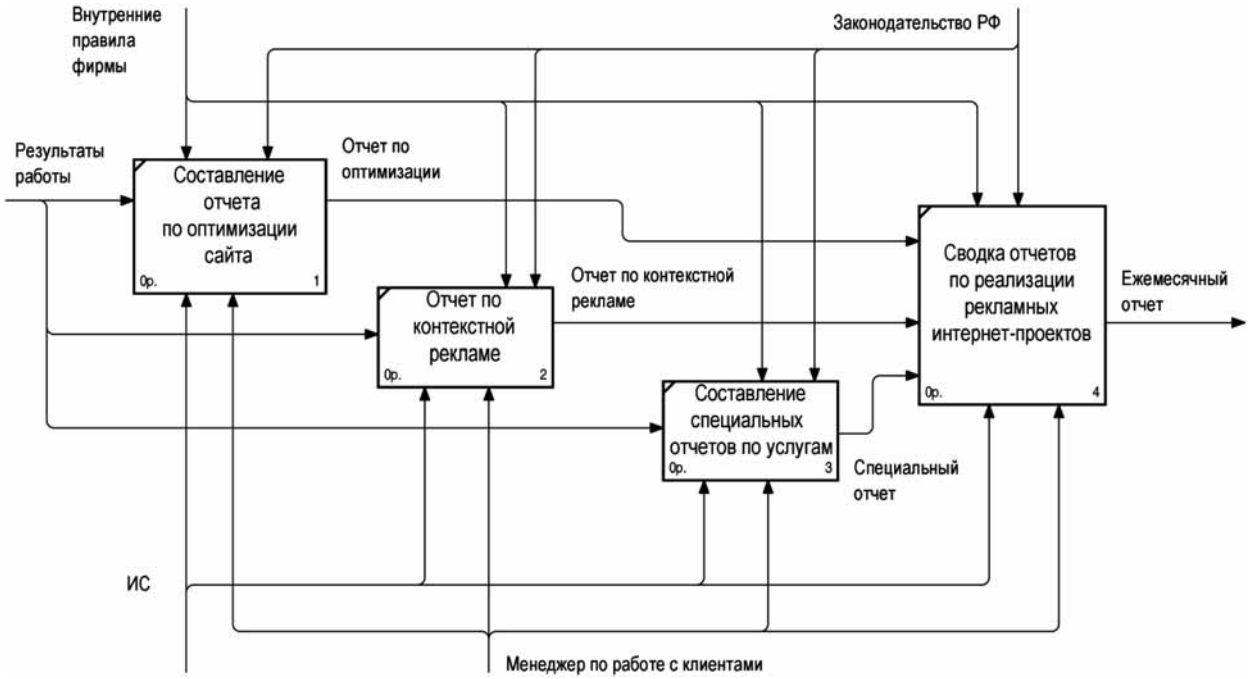


Рис. 12. Детализация процесса "Составление отчетов"

процессов составления коммерческого предложения и составления договора представлена на рис. 11.

Получив заполненную личную карту клиента, менеджер переходит к составлению коммерческого предложения. Информационная система отображает список услуг с указанными оценками маркетолога, и менеджер принимает окончательное решение о списке оказываемых услуг. Он также определяет дополнительный список услуг для клиента, которые могли бы положительно повлиять на развитие его бизнеса. После проставления итоговых оценок ИС подсчитывает стоимость проекта, на основании которой составляется коммерческое предложение. Если коммерческое предложение утверждается, менеджер с помощью ИС распечатывает договор и относит его на подпись. Если клиенту не подходит спектр услуг, предоставляемых компанией, он может отказаться от сотрудничества или попросить пересмотреть список.

После подписания договора с двух сторон, сотрудники агентства ООО "Гифтилайф" начинают проводить работы согласно договору. Детализация процесса составления отчетов о выполненных работах представлена на рис. 12.

Ежемесячно ООО "Гифтилайф" предоставляет своим клиентам отчеты по проведенным работам и положительным тенденциям развития их бизнеса. Для этого используется ИС, с помощью которой менеджер по работе с клиентами выводит отчеты на печать.

По диаграммам представленных выше бизнес-процессов можно сделать вывод, что работа менеджера по работе с клиентами кардинально не изменится. Однако появление современной ИС упростит его работу и повысит эффективность работы как менеджера, так и компании в целом. Есть все основания полагать, что ИС позволит структурировать информацию внутри компании, создать единое хранилище данных о клиентах и их заказах и снизить документооборот.

4. Личная карта клиента

Личная карта клиента, представленная в бизнес-процессах в модели TO-BE, представляет собой комплексный документ, содержащий данные заявки клиента и оценку сложности требуемых услуг. Данную карту формируют для согласования с клиентом списка услуг и их конечной стоимости. Ее заполняют и корректируют с момента принятия заказа до составления договора. Шаблоны личной карты клиента (услуги и оценки сложности их проведения, а также личные данные) представлены в табл. 1 и 2.

Благодаря структурированию данных в личной карте клиента, ИС может автоматически рассчитывать стоимость реализации рекламных интернет-проектов по формуле

$$C_{\text{общ}} = \sum_{i=1}^{n} C_{i,j},$$

где $C_{\text{общ}}$ — итоговая стоимость, руб.; n — число услуг; i — номер услуги (первый и второй столбцы анкеты, см. табл. 1, например, 1 — "Яндекс.Директ"); j — оценка, указанная для i-й услуги в анкете (третий столбец анкеты; если клиент не нуждается в услуге конкретного вида, оценка устанавливается равной 0).

Например, значение $C_{2,1}$ определяет стоимость услуги Google Adwords (оценка 1 — необходимы полномасштабные работы), а значение $C_{3,0}$ подразумевает стоимость услуги по внутренней оптимизации сайта

Таблица 1

Личная карта клиента (услуги и оценки сложности их проведения)

Номер услуги, *i*	Услуга	Оценка, *j*	Расшифровка оценки сложности проведения услуг
1	Яндекс.Директ	3	Небольшие корректировки
		2	Работы средней сложности
		1	Полномасштабные работы
2	Google Adwords	3	Небольшие корректировки
		2	Работы средней сложности
		1	Полномасштабные работы
3	Внутренняя оптимизация сайта	3	Небольшие корректировки
		2	Работы средней сложности
		1	Полномасштабные работы
4	Внешняя оптимизация сайта	3	Небольшие корректировки
		2	Работы средней сложности
		1	Полномасштабные работы
5	Баннерная реклама	2	Работы средней сложности (реклама уже существует)
		1	Полномасштабные работы (реклама отсутствует)
6	E-mail-маркетинг	3	Корректировка и настройка
		2	Полное изменение политики рассылок
		1	Полномасштабные работы (маркетинг отсутствует)
7	SMS-маркетинг	3	Корректировка и настройка
		2	Полное изменение политики рассылок
		1	Полномасштабные работы (маркетинг отсутствует)
8	Реклама в социальных сетях	3	Корректировка и настройка
		2	Полное изменение политики рекламы
		1	Полномасштабные работы (реклама отсутствует)
9	Аутсорсинг	3	Ведение деятельности без массовых корректировок
		2	Разработка и внедрение стратегии
		1	Ведение деятельности с нулевой точки развития
10	Брендинг	3	Небольшая корректировка
		2	Массовые работы по узнаваемости бренда
		1	Полномасштабные работы (брендинг отсутствует)
11	Полиграфия	3	Небольшая корректировка
		2	Разработка и создание макетов
		1	Полномасштабные работы (полиграфия отсутствует)
12	Выставочный консалтинг	1	Стандартные работы
13	Аудит деятельности	1	Стандартные работы
14	Тренинги и мастер-классы	3	Вводный курс лекций
		2	Продвинутый курс лекций
		1	Экспертный курс лекций

Таблица 2

Личная карта клиента (личные данные)

№ заявки	Уникальный номер заявки
ФИО	ФИО клиента
Контакты	Контактный телефон клиента
Сайт	URL сайта клиента
Логин и пароль к сайту	Логин и пароль к административной панели сайта
Логин и пароль Яндекс	Логин и пароль к Яндекс.Директу
Логин и пароль Google	Логин и пароль к Google Adwords

(оценка 0 — данная услуга не требуется и не будет учитываться при суммировании).

5. Анализ существующих разработок

Среди программ, предназначенных для автоматизации маркетинговой деятельности, в частности, в области интернет-маркетинга, существует большое число программных продуктов, многие из которых также предназначены для обеспечения продаж и других смежных функций. Ряд таких продуктов функционирует, как правило, в виде модулей систем, имеющих более широкие задачи. Однако можно выделить группы программ, для которых функции маркетинга первичны. К их числу относятся:

- Oracle E-Business Suite;
- SAS Marketing Automation.

По мнению авторов, Oracle E-Business Suite является наиболее полным программным комплексом глобальных бизнес-приложений, который помогает организациям принимать лучшие решения, снижать расходы и повышать производительность. Благодаря наличию большого числа межотраслевых возможностей, включающих планирование ресурсов предприятия, управление отношениями с заказчиками и планирование цепочки поставок, приложения Oracle E-Business Suite помогают клиентам управлять сложностью глобальных бизнес-сред независимо от размера организации [6].

*S*AS Marketing Automation является интегрированным решением, сочетающим в себе [7]:

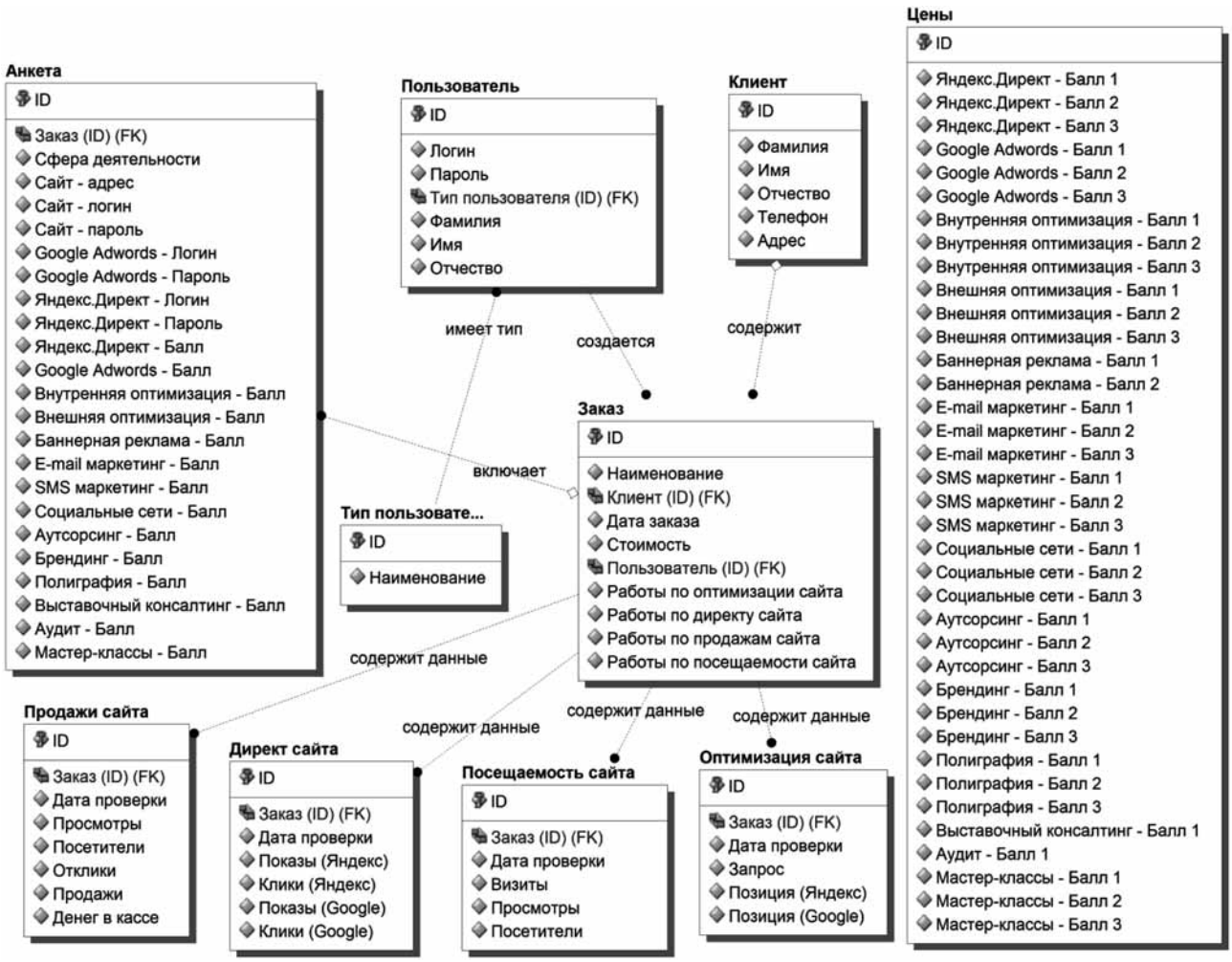


Рис. 13. Логическая диаграмма базы данных ИС компании ООО "Гифтилайф"

- платформу SAS для интеграции данных;
- предиктивную аналитику с возможностью быстро создавать и тестировать модели;
- управление кампаниями;
- отчетность для мониторинга и оценки эффективности кампаний.

Все существующие системы позволяют реализовать стандартную практику общего менеджмента, однако либо обладают излишними функциональными возможностями, либо не предоставляют опции, позволяющие автоматизировать рассмотренные выше бизнес-процессы, а создание системы на заказ является достаточно дорогой услугой для агентства. Вместе с тем требующаяся менеджеру ИС не отличается высокой сложностью, и ее разработка может быть выполнена силами IT-отдела, что и явилось окончательным решением руководства компании.

6. Технологические аспекты и общая схема работы с информационной системой ООО "Гифтилайф"

Проанализировав техническое обеспечение, а также экономические и организационные параметры компании, ИТ-отделом компании было принято решение использовать для разработки ИС язык высоко-

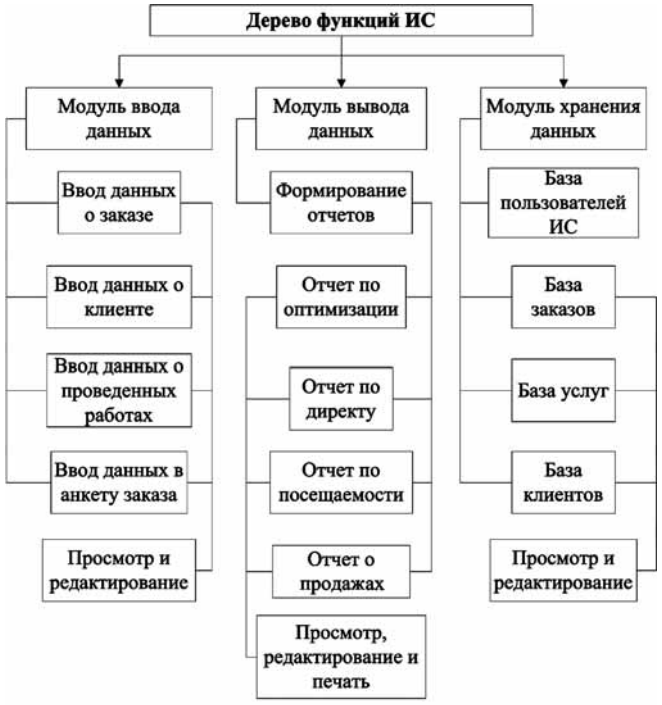


Рис. 14. Дерево функций программных модулей

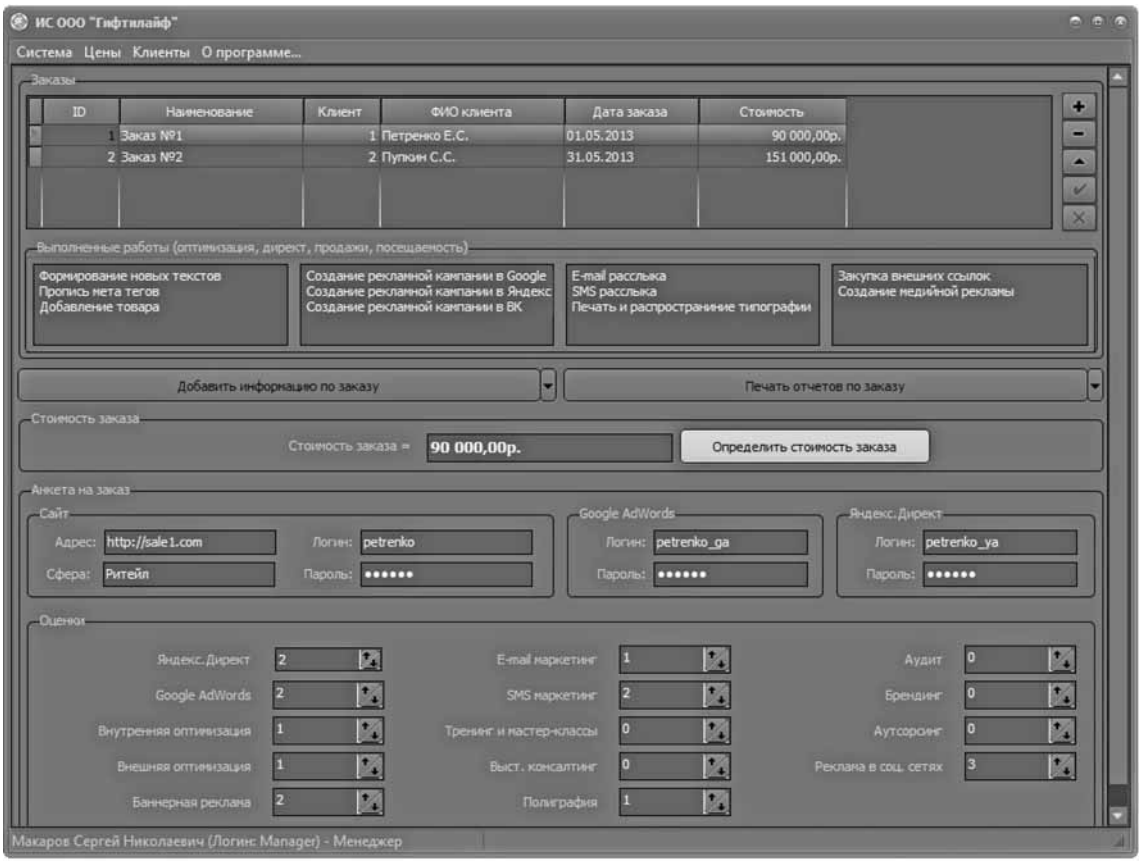


Рис. 15. Главное окно ИС

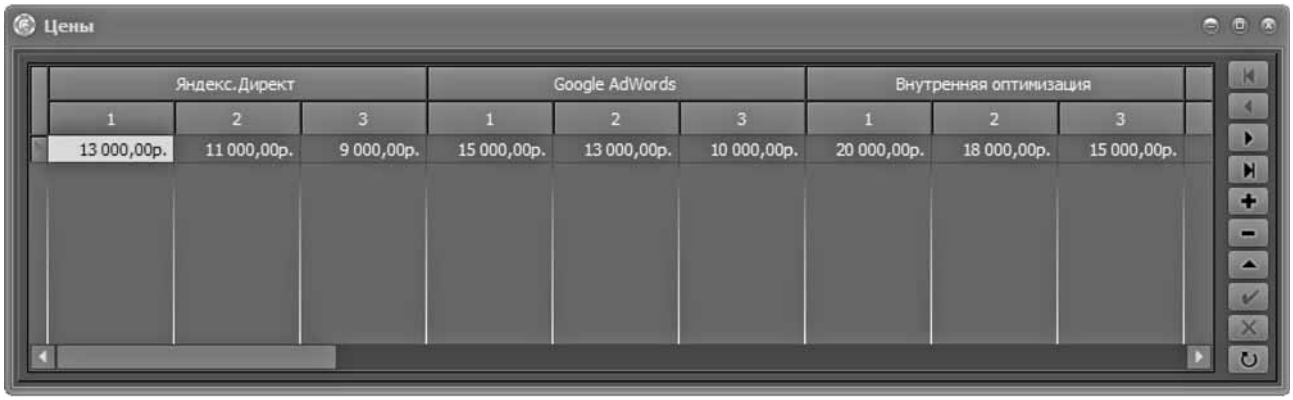
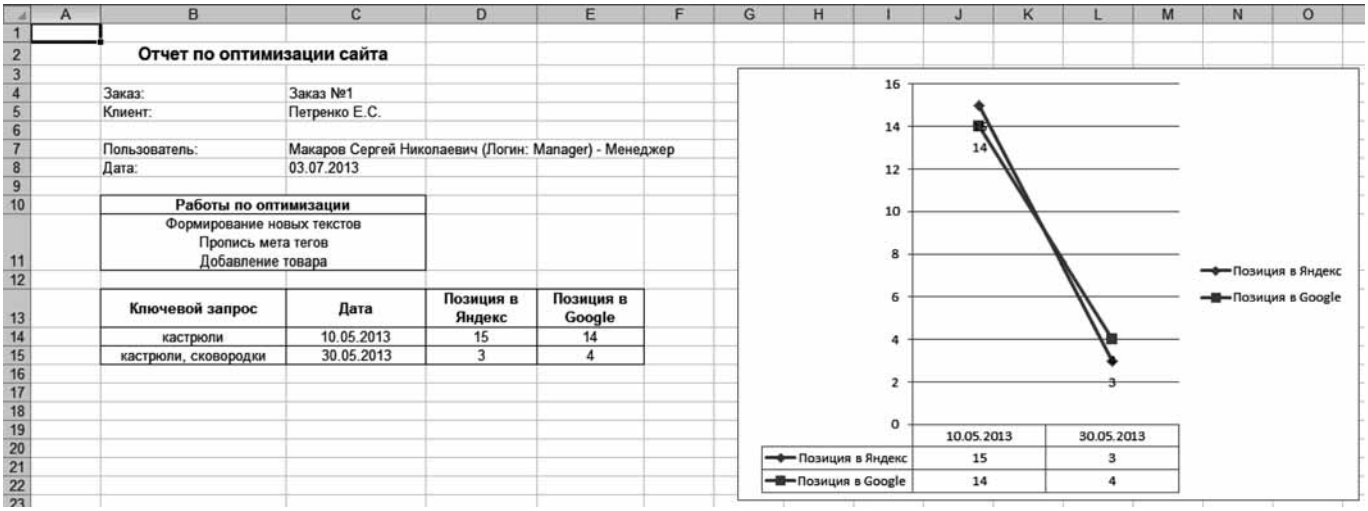


Рис. 16. Фрагмент справочника цен



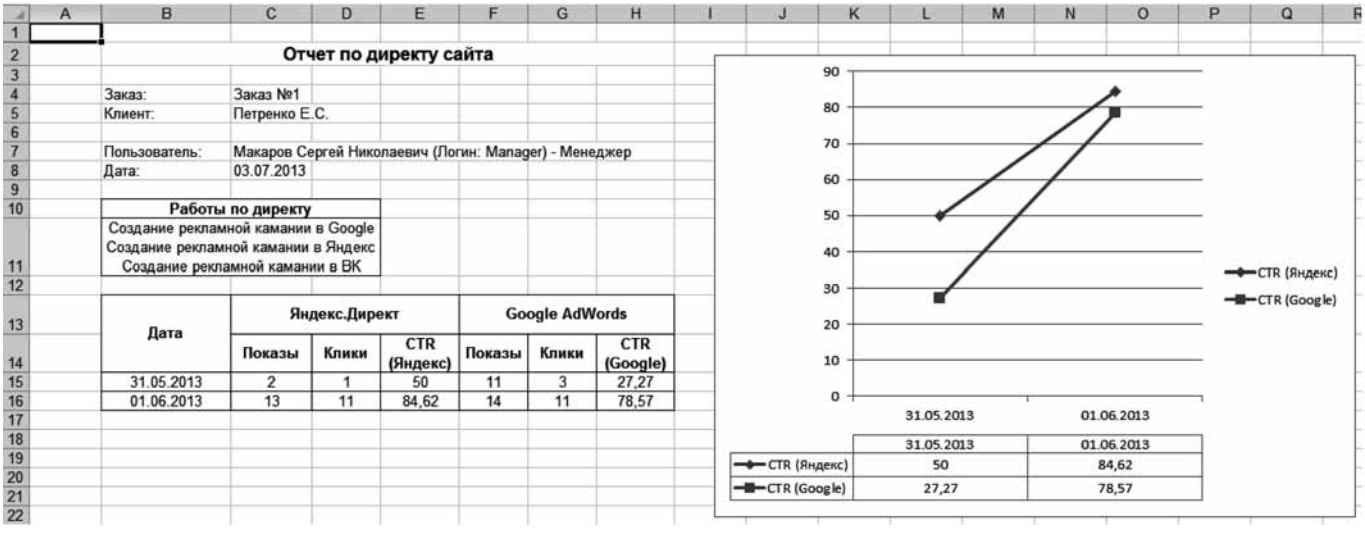
Отчет по оптимизации сайта

Заказ:	Заказ №1
Клиент:	Петренко Е.С.
Пользователь:	Макаров Сергей Николаевич (Логин: Manager) - Менеджер
Дата:	03.07.2013

Работы по оптимизации
Формирование новых текстов
Пропись мета тегов
Добавление товара

Ключевой запрос	Дата	Позиция в Яндекс	Позиция в Google
кастрюли	10.05.2013	15	14
кастрюли, сковородки	30.05.2013	3	4

Рис. 17. Пример отчета по оптимизации сайта



Отчет по директу сайта

Заказ:	Заказ №1
Клиент:	Петренко Е.С.
Пользователь:	Макаров Сергей Николаевич (Логин: Manager) - Менеджер
Дата:	03.07.2013

Работы по директу
Создание рекламной камании в Google
Создание рекламной камании в Яндекс
Создание рекламной камании в ВК

Дата	Яндекс.Директ			Google AdWords		
	Показы	Клики	CTR (Яндекс)	Показы	Клики	CTR (Google)
31.05.2013	2	1	50	11	3	27,27
01.06.2013	13	11	84,62	14	11	78,57

Рис. 18. Пример отчета по директу сайта

го уровня Delphi и СУБД Microsoft Access [8, 9]. Рассмотрим некоторые аспекты реализации ИС ООО "Гифтилайф".

На рис. 13 приведена логическая диаграмма базы данных ИС компании ООО "Гифтилайф" в нотации IDEF1X.

На рис. 14 представлено дерево функций программных модулей, используемых в разрабатываемой ИС ООО "Гифтилайф".

После установки ИС на компьютер и входа в ИС с использованием логина и пароля, менеджер попадает в главное окно приложения (рис. 15) и всю работу может осуществлять в данном окне, дополнительно используя справочники, расположенные в главном меню, — *Цены* (рис. 16) и *Клиенты*.

Работа с ИС начинается с добавления нового заказа. После этого менеджер по работе с клиентами переходит к заполнению личной карты клиента и выбирает услуги, в которых нуждается клиент (баллы проставляют согласно табл. 1 из разд. 4). Затем менеджер вносит данные о сфере деятельности заказчика, о сайте (URL, логин и пароль для доступа в административную панель), а также логины и пароли для доступа к рекламным кампаниям Яндекс и Google. На основании введенных данных маркетолог проводит анализ деятельности заказчика и устанавливает оценки по результатам своего анализа. После работы маркетолога, менеджер по работе с клиентами принимает окончательное решение по проставленным оценкам и с помощью ИС рассчитывает стоимость реализации интернет-проекта.

В разработанной ИС предусмотрено формирование договора и отчетной документации в форматах MS Excel и Word, что позволяет просматривать, редактировать и при необходимости выводить на печать созданные отчеты. На рис. 17, 18 приведены примеры отчетов по оптимизации и директу сайта.

Заключение

В статье рассмотрены особенности и результаты автоматизации деятельности по работе с клиентами в компании ООО "Гифтилайф". Представлены бизнес-процессы по работе с клиентом в моделях AS-IS и TO-BE, показаны изменения, которые вносятся с учетом внедрения ИС. Кроме того, приведена внедряемая в настоящее время личная карта клиента (анкета), а также алгоритм расчета итоговой стоимости проекта на ее основе. Также отмечены технологические особенности разработанной ИС — логическая модель базы данных, дерево функций системы. Проиллюстрирована работа приложения и приведены примеры печатных отчетов.

Список литературы

1. **Миннивалеев Ф. М.** Интернет-маркетинг как современное средство коммуникации // Актуальные проблемы экономики и права. 2011. № 3. С. 112—115.
2. **Калужский М. Л.** Инновационные формы продаж в электронной коммерции // Практический маркетинг. 2013. № 4. С. 23—34.
3. **Громов О. В.** Инструменты коммуникации интернет-маркетинга для логистических компаний // РИСК: Ресурсы, Информация, Снабжение, Конкуренция. 2010. № 2. С. 107—109.
4. **Васин А. С., Васин М. В.** Интернет-маркетинг на авторемонтных предприятиях // Финансовая аналитика: проблемы и решения. 2010. № 18. С. 55—58.
5. **Лучинская О. Ю.** Использование возможностей интернета в маркетинговой деятельности предприятия // Современные наукоемкие технологии. 2010. Т. 9. С. 71—74.
6. **Oracle** E-Business Suite. Бизнес-приложения. Oracle RU. URL: http://www.oracle.com/ru/products/applications/ebusiness/overview/index.html.
7. **Управление** маркетинговыми кампаниями | SAS Россия/СНГ. URL: http://www.sas.com/offices/europe/russia/software/solutions/campaign_management.html.
8. **Delphi XE2** Overview. URL: http://edn.embarcadero.com/article/41593.
9. **Программное** обеспечение для работы с базами данных Access — Office.com. URL: http://office.microsoft.com/ru-ru/access/.

ИНФОРМАЦИЯ

20—22 февраля 2014 г., Минск, БГУИР

IV Международная научно-техническая конференция

«Открытые семантические технологии проектирования интеллектуальных систем» (OSTIS—2014)

Основной целью ежегодных конференций OSTIS (Open Semantic Technology for Intelligent Systems) является создание условий для расширения сотрудничества различных научных школ, вузов и коммерческих организаций, направленного на разработку комплексной массовой технологии компонентного проектирования (модульного, сборочного проектирования) интеллектуальных систем.

Тематика конференции:

- Проблема независимости технологий проектирования интеллектуальных систем от различных платформ и вариантов их реализации
- Проблема интеграции интеллектуальных систем и их компонентов
- Принципы, лежащие в основе массовой технологии проектирования интеллектуальных систем на основе семантических сетей

Рабочие языки конференции: русский, белорусский, английский.
Подробности: http://conf.**ostis**.net

УДК 004.89:004.912

А. А. Харламов[1], д-р техн. наук, стар. науч. сотр., e-mail: kharlamov@analyst.ru,
Т. В. Ермоленко[2], канд. техн. наук, науч. сотр., **Г. В. Дорохина**[2], млад. науч. сотр.,
А. О. Журавлев[2], специалист,
[1] Институт высшей нервной деятельности и нейрофизиологии РАН, г. Москва
[2] Институт проблем искусственного интеллекта, г. Донецк

Предсинтаксический анализ русско-английских текстов

Приводится описание алгоритма автоматического предсинтаксического анализа текстов на русском и английском языках. Предложенный алгоритм программно реализован в системе предсинтаксического анализа, предназначенной для выделения лексических и нелексических единиц текста в целях дальнейшей их обработки синтаксическим и семантическим компонентами лингвистического процессора. Для выполнения морфологической разметки текста разработан способ представления морфологической информации словоформ английского языка в виде набора битовых полей. Разработанный алгоритм может использоваться при создании средств автоматической обработки текстов для широкого класса интеллектуальных систем.

Ключевые слова: *автоматическая обработка текста, предсинтаксический анализ, графематический анализ, морфологический анализ, базовые элементы текста*

Введение

Обработка естественного языка — достаточно перспективное направление исследований в области искусственного интеллекта. Наметившаяся в последнее время тенденция к использованию лингвистических ресурсов в системах, содержащих компоненты анализа естественно-языковых текстов, позволяет ожидать повышения качества анализа и, как следствие, более широкого использования этих ресурсов на практике. Процесс обработки текста с использованием средств и систем автоматизации включает синтаксический анализ отдельных его предложений. Однако для выполнения синтаксического анализа текст, являющийся по сути последовательностью символов, необходимо представить как последовательность предложений определенного языка. При этом предложение — это последовательность элементов, каждый из которых является либо лексической единицей в определенной грамматической форме, либо знаком препинания, либо нелексической единицей с некоторой семантической нагрузкой (например, дата, URL). Выделение и классификацию этих единиц, объединение их в предложения и абзацы необходимо выполнить до осуществления синтаксического анализа текста.

Таким образом, задача предсинтаксического анализа текста состоит в следующем.

1. Выделение лексических единиц текста с последующим установлением их принадлежности к определенному языку и нахождение грамматических характеристик этих единиц путем морфологического анализа.

2. Выделение нелексических единиц и определение их семантической нагрузки.

3. Разделение текста на предложения, абзацы, выделение структурных элементов текста (например, заголовков, названий таблиц, подписей под рисунками, диаграммами и схемами).

Алгоритм предсинтаксического анализа текстов, как всякий лингвистический алгоритм, при всей своей кажущейся простоте содержит много непредвиденных сложностей. Уже при разбиении текста на предложения возникает ряд трудностей, связанных с неоднозначной расстановкой точек, поскольку точки могут относиться, помимо прочего, и к различного рода сокращениям. Восклицательный и вопросительный знаки часто используют в выразительных вставках в тексте

и, следовательно, знаки препинания не являются стопроцентной гарантией окончания предложения.

Ниже изложен алгоритм предсинтаксического анализа русско-английских текстов, с помощью которого удается получать базовые структурные единицы текста с их описанием в виде, удобном для дальнейшей обработки синтаксическим и семантическим компонентами лингвистического процессора.

Общая схема алгоритма предсинтаксического анализа

Известным программным продуктом группы АОТ [1], в котором реализован полный лингвистический анализ текста, включая его досинтаксическую обработку, является программная система ДИАЛИНГ [2]. Цепочку анализаторов текста, реализующих последовательно графематический, морфологический, синтаксический и, частично, семантический анализ, разработанных на базе системы ДИАЛИНГ, используют крупные компании для создания коммерческих программных продуктов с элементами автоматической обработки текстов. В системе ДИАЛИНГ до синтаксического анализа программно реализованы следующие этапы анализа: графематический анализ, морфологический анализ, постморфологический анализ и фрагментация. При этом в задачу графематического анализа входят: разделение входного текста на слова, разделители и другие структурные элементы; сборка слов, написанных в разрядку; выделение устойчивых оборотов, не имеющих словоизменительных вариантов; выделение ФИО, когда имя и отчество написаны инициалами; выделение электронных адресов и имен файлов; сегментация входного текста на предложения; выделение абзацев, заголовков, примечаний. На этапе постморфологического анализа упомянутой работы к результатам предшествующих этапов применяют несколько правил, направленных на снижение неоднозначности результатов морфологического анализа.

Отличием алгоритма предсинтаксической обработки текста, предложенного в настоящей работе, от реализованного в системе ДИАЛИНГ, является то, что морфологическая разметка выполняется до графематического анализа. Кроме того, для описания базовых элементов в реализации этого алгоритма используется оригинальная система дескрипторов.

Схема выполнения предсинтаксического анализа представлена на рис. 1.

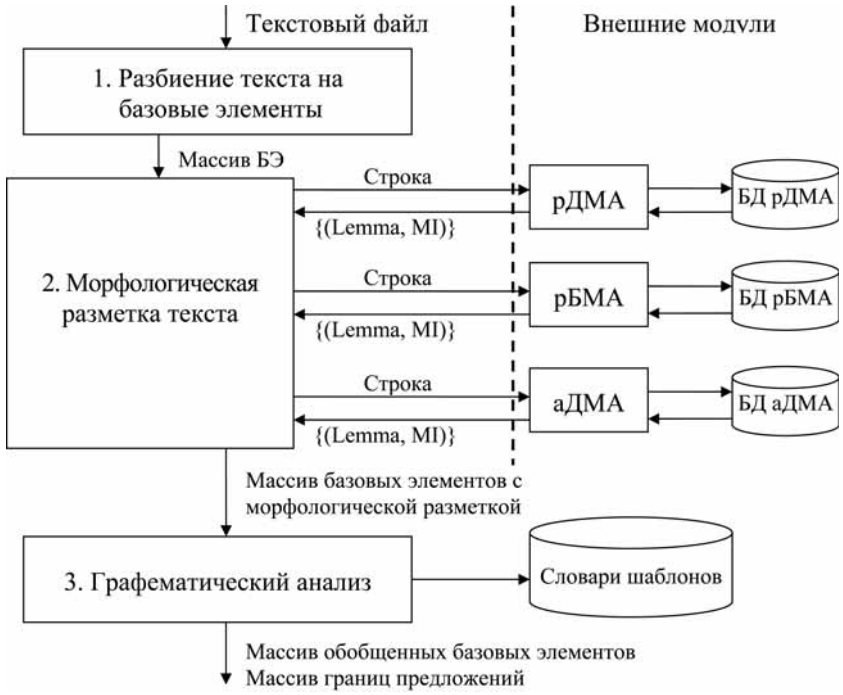


Рис. 1. Схема выполнения предсинтаксического анализа

Этап разбиения текста на базовые элементы предполагает выделение в тексте последовательностей символов одного алфавита (алфавит символов кириллицы, алфавит символов латиницы, алфавит знаков препинания, алфавит разделителей, алфавит для записи числовых значений, алфавит скобок и кавычек, все остальные символы). Такие последовательности символов называют базовыми элементами (БЭ).

На этапе морфологической разметки текста выполняется морфологический анализ написаний базовых элементов, состоящих из символов латиницы или кириллицы. При этом используют программные модули, реализующие:

— декларативный морфологический анализ слов русского языка [3] (блок рДМА на рис. 1);

— бессловарный морфологический анализ слов русского языка [4] (блок рБМА на рис. 1);

— декларативный морфологический анализ слов английского языка (блок аДМА на рис. 1).

Этап графематического анализа позволяет интерпретировать базовые элементы и последовательности базовых элементов как нелексические единицы с определенной семантической нагрузкой (e-mail, URL, дата, имя файла и подобные им) или лексические единства, выступающие в предложении как единое целое (фамилия с инициалами; фамилия с предшествующим ей сокращением названия титула или названия нетитулованного лица; составное географическое название; географическое или административное название с предшествующим ему сокращением).

Еще одной задачей, которая решается на этапе графематического анализа, является разбиение текста на предложения.

Обработанный описанным выше способом текст готов к дальнейшей обработке синтаксическим компонентом лингвистического процессора. Это означает, что он разбит на предложения, каждое из которых представлено в виде последовательности лексических, нелексических элементов и лексических единств. При этом для лексических элементов выполнена морфологическая разметка, а для нелексических элементов и лексических единств определен их семантический класс.

На основе разработанного алгоритма создана программная система предсинтаксического анализа (далее для краткости — система), каждый этап обработки в которой реализован в соответствующем ее модуле. Остановимся на программной реализации каждого из этапов предсинтаксического анализа более подробно.

Выделение в тексте базовых элементов

Первый этап предсинтаксического анализа реализован в модуле выделения базовых элементов, в результате функционирования которого формируется внутреннее представление текста. Это представление, как показано далее, будет дополняться на последующих этапах. Каждый базовый элемент описывается структурой, поля которой представлены в табл. 1.

На этом этапе для базового элемента фиксируются:

— алфавит, из символов которого он состоит;

— регистр (нижний, верхний, заголовочный — первый символ слова находится в верхнем регистре, остальные — в нижнем; смешанный — любая другая комбинация регистров);

— флаг, который указывает, следует ли за базовым элементом разделитель (пробел, табуляция, перевод каретки) или последовательность разделителей;

Таблица 1

Поля структуры, описывающей базовый элемент

№	Имя	Описание
1	WordLow	Последовательность символов базового элемента, приведенная для последовательностей символов русского или английского алфавита к нижнему регистру
2	WordRegistr	Беззнаковое целое число (32 бита). Двоичная запись этого числа показывает, в каком регистре находится каждый символ последовательности (1 — заглавный, 0 — строчный, данные о первом символе хранятся в нулевом бите, о втором — в первом и т.д.). Определяется только для последовательностей символов русского или английского алфавита. Приведем пример, заметив, что биты в двоичной записи числа принято нумеровать справа-налево. Так, в последовательности символов *MaMa* первый и третий символы — заглавные, остальные — строчные. Значит этой строке соответствует двоичное значение, в котором нулевой и второй биты равны единице, а первый и третий — нулю. Так как строка *MaMa* состоит из четырех символов, то в поле WordRegistr будут значимы четыре младших бита, а остальные будут заполнены незначащими нулями. Таким образом, в двоичной системе счисления $(\text{WordRegistr})_2$=0101, что в десятеричной системе счисления соответствует числу $5= 2^0+2^2$
3	WordLen	Длина слова (число символов)
4	WordBegin	Смещение слова от начала текста (число символов)
5	Alph_Num	Число, задающее принадлежность к алфавиту: 0 — последовательность символов не является ни словом, ни знаком препинания, ни разделителем между словами; 1 — слово русского языка; 2 − слово английского языка; 3 − последовательность цифр; 4 − знак препинания; 5 — скобки; 6 — недостаточная длина слова; 7 − сокращения, аббревиатуры; 8 — цифра + буква; 9 — разделитель между словами
6	Reg_Num	Значение регистра последовательности символов: 1 — начинается с прописной; 2 — начинается с заглавной; 3 − начинается с заглавной, остальные прописные; 4 − начинается с прописной, остальные заглавные; 5 − все прописные; 6 − заглавные; 7 − вперемешку заглавные и прописные.
7	IsSokrAbb	Флаг, принимающий значение 1, если базовый элемент является потенциальной аббревиатурой
8	IsSentEnd	Флаг, принимающий значение 1, если слово является концом предложения
9	IsParagrafEnd	Флаг, принимающий значение 1, если за словом следует символ перевода каретки или разрыв строки
10	IsDot	Флаг, принимающий значение 1, если базовый элемент является точкой
11	IsDeviderAfter	Флаг, принимающий значение 1, если есть разделитель между текущим словом и последующим (пробел, табуляция, перевод каретки)
12	KindOfBracket	Вид скобочки (1 — круглая, 2 — квадратная, 3 — фигурная,4 — угловая, 5 — одинарная кавычка, 6 — двойная кавычка, 7 — иная), если это не скобочка — 0
13	BracketType	Тип скобочки (1 — открывающая, 2 — закрывающая, 3 — нейтральная)
14	lemm_s	Массив переменной длины, хранящий элементы типа *string*. Заполняется на этапе морфологической разметки текста леммами слова, имеющего написание WordLow. Число элементов в массиве равно числу найденных системой морфологических интерпретаций (различных сочетаний леммы и морфологической информации) слова
15	Mi_s	Массив переменной длины, хранящий элементы типа беззнаковое целое. Заполняется на этапе морфологической разметки текста морфологической информацией (см. [3]) слова, имеющего написание WordLow. Число элементов в массивах lemm_s и Mi_s совпадает. Элементы этих массивов с одинаковым индексом описывают одну морфологическую интерпретацию слова

— флаг, который указывает, следует ли за базовым элементом знак препинания, завершающий предложение;

— флаг, указывающий, следует ли за базовым элементом перевод каретки;

— флаг, указывающий, является ли базовый элемент точкой;

— флаг, указывающий, является ли базовый элемент скобкой.

Написание каждого базового элемента система заносит в хранилище строк [5], запоминая его позицию в тексте. Такая организация позволяет по окончании этапа выделения базовых элементов определять число употреблений в тексте каждого базового элемента. Написание базового элемента, являющегося последовательностью символов латиницы или кириллицы, система также приводит к нижнему регистру и сохраняет полученную строку в отдельном хранилище строк, запоминая его позицию в тексте. Сохраняется также информация, позволяющая учитывать число употреблений строки в определенном регистре. Таким образом, реализуется простой способ выделения слов, которые являются потенциальными именами собственными, аббревиатурами.

В результате работы модуля выделения в тексте базовых элементов система создает массив таких элементов с их дескрипторами, которые необходимы для классификации базовых элементов и определения границ предложения. Написания базовых элементов, состоящих из символов латиницы и кириллицы, система подвергает морфологическому анализу, результаты которого заносятся в поля lemm_s и Mi_s (табл. 1).

Морфологическая разметка текста

На рис. 2 приведена блок-схема алгоритма морфологического анализа слова русского языка (параметр word на блок-схеме), который реализован в модуле морфологической разметки текста. Морфологический анализ начинается с процедуры приведения слова к нижнему регистру (ConvertToLow) и выполняется путем обращения к процедурам внешних модулей:

— декларативного морфологического анализа (НайтиЛемму) [3];

— бессловарного морфологического анализа (НайтиЛеммуБМА) [4];

— морфологического анализа слов с дефисом (Найти лемму с учетом дефиса) [6].

Результат морфологического анализа система сохраняет в полях lemm_s и Mi_s структуры, описывающей базовый элемент.

Для анализа слов английского языка была разработана процедура декларативного морфологического анализа слов английского языка. Опишем используемый в ней способ представления морфологической информации. Морфологическая информация хранится в виде набора битовых полей, что отвечает требованиям компактности, однозначности и простоты извлечения отдельных морфологических характеристик словоформы. Таблица 2 содержит перечень обозначений с помощью чисел и макроопределений, которые используются в системе для задания морфологической информации слова английского языка. Эти обозначения подобраны так, чтобы совпадали одинаковые значения одних и тех же категорий для русского и английского языков. В столбце "Совпадает с русским" такие обозначения помечены символом "+". Морфологическая информация словоформы формируется применением побитового "или", например _Noun_en | _Nominative_en | _Singular_en.

Значение определенной грамматической категории для слова по его морфологической информации находится с помощью масок категорий (табл. 3).

Применив операцию побитового "и" к значению морфологической информации словоформы и маски

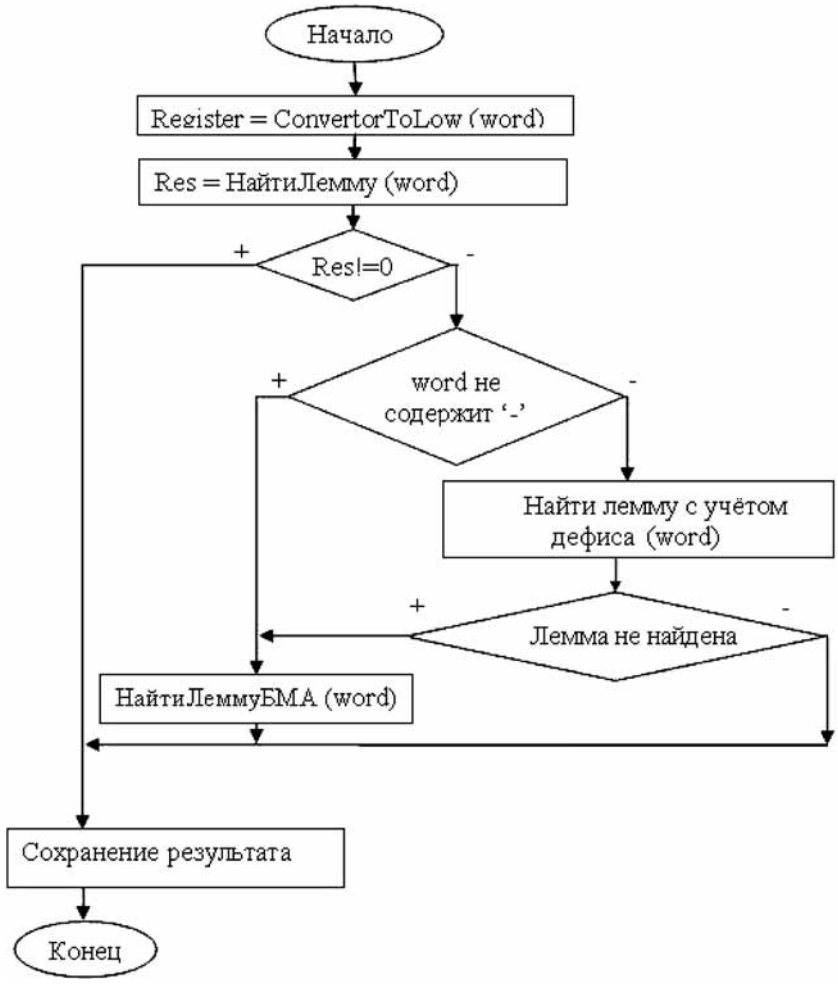


Рис. 2. Алгоритм морфологического анализа слов русского языка

Таблица 2

Значения грамматических категорий для английского языка

Обозначение в программе		Грамматические категории		Совпадает с русским
Число	Макроопределение	Категория	Значение	
0x00000001	_Nominative_en	Падеж	Именительный	+
0x00000002	_Prityag_en		Притяжательный	–
0x00000003	_Objekt_en		Объектный падеж	–
0x00000004	_PritagAbsol_en		Притяжательный абсолютный	–
0x00000008	_Masculine_en	Род	Мужской	+
0x00000010	_Feminine_n		Женский	+
0x00000018	_Neuter_en		Средний	+
0x00000020	_Singular_en	Число	Единственное	+
0x00000040	_Plural_en		Множественное	+
0x00000080	_Pres_en	Время	Настоящее	+
0x00000100	_Future_en		Будущее	+
0x00000180	_Past_en		Прошедшее	+
0x00000200	_FaceFir_en	Лицо	Первое	+
0x00000400	_FaceSec_en		Второе	+
0x00000600	_FaceThi_en		Третье	+
0x00000800	_Active_en	Залог	Действительный	+
0x00001000	_Passive_en		Страдательный	+
0x00002000	_ComparativeFormOfAdj_en	Степень сравнения	Сравнительная	+
0x00004000	_ExellentFormOfAdj_en		Превосходная	+
0x00008000	_Verb_en	Часть речи	Глагол	+
0x00010000	_Participle_en		Причастие	+
0x00018000	_Gerund_en		Деепричастие	+
0x00020000	_Adjective_en		Прилагательное	+
0x00028000	_Noun_en		Существительное	+
0x00030000	_Pronoun_en		Местоимение	+
0x00038000	_Numeral_en		Числительное	+
0x00040000	_AdVerb_en		Наречие	+
0x00048000	_Preposition_en		Предлог	+
0x00050000	_Conjunction_en		Союз	+
0x00058000	_Particle_en		Частица	+
0x00060000	_Interjection_en		Междометие	+
0x00070000	_Article_en		Артикль	–
0x00078000	_ComparativeWord_en		Сравнительное слово	+
0x00080000	_Animate_en	Одушевленность	Одушевленное	+
0x00100000	_NotAnimate_en		Неодушевленное	+
0x00200000	_ReturnPron_en		Возвратно-усилительное местоимение	–
0x00400000	_2st_Verb_form_en	Форма глагола	Прошедшее неопределенное время действительного залога	–
0x00800000	_3st_Verb_form_en		Причастие прошедшего времени	–
0x00C00000	_4st_Verb_form_en		Причастие настоящего времени	–
0x01000000	_Count_en	Тип числительного	Количественное	+
0x02000000	_Ordinal_en		Порядковое	+
0x04000000	_DefiniteArt_en	Тип артикля	Неопределенный	–
0x08000000	_IndefiniteArt_en		Определенный	–
0x10000000	_IndefiniteT_en	Группы времен	Простое	–
0x20000000	_Continuous_en		Длительное	–
0x30000000	_Perfect_en		Совершенное	–
0x40000000	_PerfectContinuous_en		Совершенное длительное	–

Таблица 3

Маски категорий морфологической информации

Числовое значение	Макроопределение	Маска категории
0x00000007	case_mask_en	Падеж
0x00000018	rod_mask_en	Род
0x00000060	count_mask_en	Число
0x00000180	time_mask_en	Время
0x00000600	face_mask_en	Лицо
0x00001800	active_passive_mask_en	Залог
0x00006000	adjfrm_mask_en	Степень сравнения, краткость
0x00078000	part_of_speech_mask_en	Часть речи
0x00180000	animate_mask_en	Одушевленность
0x00C00000	aspect_of_verb_mask_en	Вид глагола
0x03000000	number_type_mask_en	Тип числительного
0x0C000000	article_type_mask_en	Тип артикля
0x70000000	tence_group_mask_en	Группы времен

определенной категории, можем получить значение этой грамматической категории для словоформы. Если категория соответствует словоформе, получается ненулевое значение. Например, определение значения категории числа происходит путем применения операции побитового "и" к значению морфологической информации и маски категории. Если словоформе категория не соответствует, то результат этой операции равен 0. Ниже приведен пример определения значения категории "число" для слова, морфологическая информация которого хранится в переменной MI:

MI & count_mask.

Результат: _Singular_en, _Plural_en или 0.

Все словарные формы, включенные в парадигму, состоят из одного слова и отличаются друг от друга по написанию. Из одной словоформы состоят парадигмы следующих частей речи: наречие, союз, междометие, предлог. Примеры парадигм остальных частей речи приведены в табл. 4.

Каждая словоформа парадигмы описывается тремя значениями: написанием; пометкой, указывающей, является словоформа леммой (1) или нет (0); значением морфологической информации как совокупности зна-

Таблица 4

Примеры парадигм изменяемых частей речи английского языка

Часть речи	Написание	Лемма, 1/0	Морфологическая информация
Прилагательное	angry	1	_Adjective_en
	angrier	0	_Adjective_en | _ComparativeFormOfAdj_en
	angriest	0	_Adjective_en | _ExellentFormOfAdj_en
Существительное	project	1	_Noun_en | _Singular_en
	project's	0	_Noun_en | _Singular_en | _Prityag_en
	projects	0	_Noun_en | _Plural_en
	projects'	0	_Noun_en | _Plural_en | _Prityag_en
Местоимение	you	1	_Pronoun_en | _Singular_en | _FaceSec_en | _Nominative_en
	you	0	_Pronoun_en | _Singular_en | _FaceSec_en | _Objekt_en
	your	0	_Pronoun_en | _Singular_en | _FaceSec_en | _Prityag_en
	yours	0	_Pronoun_en | _Singular_en | _FaceSec_en | _PritagAbsol_en
	yourself	0	_Pronoun_en | _Singular_en | _FaceSec_en | _ReturnPron_en
	you	1	_Pronoun_en | _Plural_en | _FaceSec_en | _Nominative_en
	you	0	_Pronoun_en | _Plural_en | _FaceSec_en | _Objekt_en
	your	0	_Pronoun_en | _Plural_en | _FaceSec_en | _Prityag_en
	yours	0	_Pronoun_en | _Plural_en | _FaceSec_en | _PritagAbsol_en
	yourselves	0	_Pronoun_en | _Plural_en | _FaceSec_en | _ReturnPron_en
	I	1	_Pronoun_en | _Singular_en | _FaceFir_en | _Nominative_en
	me	0	_Pronoun_en | _Singular_en | _FaceFir_en | _Objekt_en
	my	0	_Pronoun_en | _Singular_en | _FaceFir_en | _Prityag_en
	mine	0	_Pronoun_en | _Singular_en | _FaceFir_en | _PritagAbsol_en
	myself	0	_Pronoun_en | _Singular_en | _FaceFir_en | _ReturnPron_en
Глагол	go	1	_Verb_en
	went	0	_Verb_en | _2st_Verb_form_en
	gone	0	_Verb_en | _3st_Verb_form_en
	going	0	_Verb_en | _4st_Verb_form_en
	goes	0	_Verb_en |_Pres_en | _IndefiniteT_en | _FaceThi_en | _Singular_en

чений отдельных грамматических категорий, объединенных операцией побитового "или" (в табл. 4 обозначена символом '|').

Только для местоимений используют следующие значения грамматической категории "падеж": "именительный", "объектный падеж", "притяжательный абсолютный".

Графематический анализ

Базовый элемент не всегда соответствует отдельной смысловой единице текста. Причина может заключаться в том, что отдельные виды используемых в текстах знаков состоят из символов нескольких алфавитов, а значит по используемому системой алгоритму будут считаться разными базовыми элементами. Примером могут служить адреса электронной почты или смайлики. Сокращения, оканчивающиеся точкой, также нецелесообразно рассматривать как два знака — неизвестное системе анализа текста слово и знак препинания "точка". Некоторые последовательности базовых элементов, отделенные друг от друга разделителем (пробел, табуляция, перевод каретки) или последовательностью разделителей, также целесообразно объединить в один смысловой элемент. Например, дата, фамилия с инициалами, фамилия с предшествующим сокращением названия титула.

Такие последовательности базовых элементов будем объединять в единицы, которые назовем обобщенными базовыми элементами (ОБЭ). Они представляют собой сегменты текста, в табл. 5 приведено описание их структуры.

Если ОБЭ состоит из одного базового элемента, то он и является сегментом, следовательно, все три первых поля (WBegin, WEnd и Wchief) имеют одно и то же значение. Если ОБЭ состоит из нескольких базовых элементов, но ни один из них не описывает весь ОБЭ (например, электронный адрес), то поле Wchief принимает значение 0. Поле SegmType принимает значение из множества объявленных в системе констант: _GroupFIO, _GroupNameSername, _Smile, _PhoneNumber, _Initial, _e_mail, _IP, _URL, _DataTime, _Time, _Reduction, _FileName, _KeyComb или 0. Поле MI — морфологическая информация, описывающая поведение обобщенного базового элемента в предложении.

Таблица 5

Поля структуры для описания обобщенного базового элемента

Имя элемента	Тип	Значение
WBegin	unsigned int	Номер базового элемента начала сегмента
WEnd	unsigned int	Номер базового элемента конца сегмента
Wchief	unsigned int	Номер базового элемента главного слова в сегменте
WordIV	vector <unsigned int>	Массив номеров вариантов интерпретации главного слова, используется, когда Wchief<>0
SegmType	eSegmType	Константа, указывающая на тип сегмента, если ОБЭ является сегментом, получаемым по шаблону или словарю
MI	UINT	В случае, когда главное слово в ОБЭ не имеет морфологической информации, можно использовать это поле для записи в него альтернативной морфологической информации

Графематичский анализ, алгоритм которого приведен далее, выполняется в три шага. На первом шаге происходит объединение написаний базовых элементов, между которыми нет разделителей, и проверка принадлежности полученной строки шаблонам элементов следующих видов: смайл, адрес электронной почты, URL, IP-адрес, дата, номер телефона, инициал, сокращение, имя файла, сочетание клавиш. На втором шаге анализируются последовательности ОБЭ на предмет соответствия шаблонам. На третьем шаге текст разбивается на предложения.

Алгоритм графематического анализа

На вход подается список слов во внутреннем представлении.

Входные данные: *v* — массив структур, описывающих БЭ текста.

Выходные данные: *gw* — массив структур, описывающих ОБЭ текста.

Константы: *N* — множество типов сегментов, которым может быть ОБЭ.

N = {_GroupFIO, _GroupNameSername, _Smile, _PhoneNumber, _Initial, _e_mail, _IP, _URL, _DataTime, _Time, _Reduction, _FileName, _KeyComb }

Начало

1. ***Для*** всех БЭ

1.1. Выделение группы неразделенных БЭ.

1.2. Формирование строки *word_g* — объединенной группы неразделенных БЭ.

1.3. Поиск шаблона, которому может соответствовать *word_g* и, в случае успешного поиска, присвоение полю SegType значения, соответствующего шаблону.

1.4. ***Если*** шаблон найден,

1.4.1. ***то***

1.4.1.1. Создание одного ОБЭ — *ggw*.

1.4.1.2. Заполнение полей этой структуры: Wbegin — номер первого БЭ, входящего в неразделенный сегмент, WEnd — номер последнего БЭ этого сегмента, SegmType — значение переменной SegType;

1.4.1.3. ***Если*** в ОБЭ вошли БЭ, которые могут являться концом предложения,

то корректировка этих БЭ и массива номеров БЭ, потенциально являющихся концом предложения.

1.4.1.4. Добавление созданного ОБЭ в конец массива *gw*.

1.4.2. ***иначе*** последовательно для каждого БЭ этого сегмента создание отдельного ОБЭ и добавление его в конец массива *gw*.

2. ***Для*** $i := 1$ ***до*** $i < gw.size()$, шаг 1.

2.1. $j := i + 1$, $f := 0$, $St = 0$.

2.2. ***Пока*** $f \neq 0$ ***ИЛИ*** $j = i + 1$

2.2.1.Формирование строки *word_g* путем объединения написаний ОБЭ с номерами из диапазона [*i*, *j*].

2.2.2. *St* := *f*.

2.2.3. *f* := ПроверкаСоответствияШаблону (word_g).

2.2.4. *j*++;

2.3. ***Если*** *St* = 0 ***И*** *St St* ∈ *N*

2.3.1. ***то***

2.3.1.1. Создание ОБЭ — *ggw*

2.3.1.2. *ggw*.Wbegin := gw[*i*].Wbegin, *ggw*.Wend := *gw*[*j* − 1].Wend, *ggw*.SegType := *St*

2.3.1.3. ***Если*** в *ggw* вошли БЭ, которые могут являться концом предложения,

то корректировка этих БЭ и массива номеров БЭ, потенциально являющихся концом предложения.

2.3.1.4. Удаление из массива *gw* элементов с номерами из диапазона [*i*, *j* − 1].

2.3.1.5. Вставка созданного ОБЭ *ggw* в массив *gw* на *i*-ю позицию.

3. Разбиение текста на предложения.

Конец

Проверка принадлежности строки ко множествам смайлов, дат, телефонных номеров и т. п. выполняется с помощью поиска по словарям соответствующих шаблонов. Опишем используемые шаблоны более подробно.

Как известно, электронный адрес содержит две части: первая, местная часть адреса или имя пользователя, расположена перед знаком @; вторая, доменная часть, следует за знаком @. Принадлежность строки множеству электронных адресов проверяется по наличию внутри нее символа @ и соответствию частей адреса ограничениям, накладываемым на них форматом email-адресов. Так, местная часть адреса содержит не более 64 символов и может состоять из следующих символов: заглавные и строчные английские буквы (a—z, A—Z); цифры 0...9; символ *точка* при условии, что это не первый или последний символ, а также не встречается два или больше раза подряд (например. John..Doe@example.com); специальные символы из множества {‘!’, ‘#’, ‘$’, ‘ %’, ‘&’, ‘’’, ‘*’, ‘+’, ‘–‘, ‘/’, ‘=’, ‘?’, ‘^’, ‘_’, ‘{’, ‘|’, ‘}’, ‘~’}. Доменная часть — это строка символов, оканчивающаяся доменным именем верхнего уровня.

Шаблоны URL приведены в табл. 6. При этом использованы следующие обозначения: S — схема обращения к ресурсу (*ftp*, *http*, *rtsp, https, gopher, mailto, news, nntp, irc, prospero, telnet, wais, xmpp, file, data, tel*); U — логин (имя пользователя, используемое для доступа к ресурсу); P — пароль указанного пользователя; H — хост (доменное имя хоста); G — порт хоста для подключения; R — URL-путь (уточняющая информация о месте нахождения ресурса).

Таблица 6

Шаблоны URL

S://U:P@H:G/R	S://@H/R	U:P@H:G/R	H/R
S://U:@H:G/R	S://H/R	U:@H:G/R	U:P@H
S://@H:G/R	S://U:P@H	H:G/R	U:@H
S://H:G/R	S://U:@H	U:P@H/R	H
S://U:P@H/R	S://@H	U:@H/R	
S://U:@H/R	S://H	@H/R	

При проверке принадлежности строки множеству IP-адресов система использует спецификации стандарта IPv4.

В системе использованы шаблоны даты и времени (табл. 7), в которых применяются такие обозначения: N — сокращенное название дня недели; D — цифра, день даты, C — сокращенное название месяца; Y — цифра, года; S — полное название месяца; R — полное название дня недели; M — цифра, месяц; H — цифра, часы; V — цифра, минуты; K — цифра, секунды.

Таблица 7

Шаблоны даты и времени

MM.YY	D.MM.YYYY	DD/MM/YYYY	D C. YY	R D S, YYYY
DD.C	DD.M.YYYY	DD-MM-YY	D S, YYYY	R DD S, YYYY
HH:VV	DD.MM.YY	DD-MM-YYYY	D. S YYYY	DD.MM.YY HH:VV
MM-DD	DD.MM.YYYY	YY-MM-DD	D S, YY	HH:VV:KK
D.M.YY	D/M/YYYY	YYYY-MM-DD	D S YY	HH:VV AM/PM
D.MM.YY	D/MM/YYYY	D C, YY	N, D C, YY	HH:VV:KK AM/PM
DD.M.YY	DD/M/YYYY	D C, YYYY	N DD.C YY	DD.MM.YYYY HH:VV:KK
D.M.YYYY	DD/MM/YY	D. C. YYYY	N, D S, YYYY	

Словари шаблонов хранятся в текстовых файлах и загружаются системой перед началом работы. Такая организация позволяет дополнять и изменять перечень известных системе шаблонов.

По окончании формирования массива ОБЭ система разбивает обрабатываемый текст на сегменты, соответствующие отдельным предложениям. Результат сохраняется в бинарном файле. Впоследствии из этого файла можно загружать (в нем сохранять) весь текст или только несколько предложений с номерами, принадлежащими заданному пользователем диапазону. Это позволяет выполнять синтаксический анализ текста в описанном представлении несколькими вычислительными устройствами параллельно.

Пример работы системы

Рассмотрим пример обработки текста системой предсинтаксического анализа. Подадим на вход системы текст из четырех предложений, включающий два предложения на английском языке и два на русском языке:

Сегодня 25-05-2010 в 16:00 туман накрыл города, села, деревни и т. п. Мерзко и зябко! According to the site http://ru.wikipedia.org/wiki/Hu_Jintao, Hu Jintao is the President of the People's Republic of China. His email is hujin@mail.ch.

Приведем значения отдельных полей структуры, описывающей базовые элементы текста в момент окончания разбиения текста на базовые элементы (табл. 8).

Результаты морфологического анализа лексических единиц текста, присутствующих в морфологической базе данных системы, даны в табл. 9.

В табл. 9 при описании морфологической информации слов русского языка ряд обозначений аналогичен

Таблица 8

Базовые элементы текста

№ элемента	Базовый элемент	WordRegistr	Alph_Num	IsSentEnd	IsDot	IsDeviderAfter	№ элемента	Базовый элемент	WordRegistr	Alph_Num	IsSentEnd	IsDot	IsDeviderAfter
1	сегодня	1	1	0	0	1	35	ru	0	2	0	0	0
2	25	0	3	0	0	0	36	.	0	4	0	1	0
3	—	0	4	0	0	0	37	wikipedia	0	2	0	0	0
4	05	0	3	0	0	0	38	.	0	4	0	1	0
5	—	0	4	0	0	0	39	org	0	2	0	0	0
6	2010	0	3	0	0	1	40	/	0	5	0	0	0
7	в	0	1	0	0	1	41	wiki	0	2	0	0	0
8	16	0	3	0	0	0	42	/	0	5	0	0	0
9	:	0	4	0	0	0	43	hu	1	2	0	0	0
10	00	0	3	0	0	1	44	_	0	0	0	0	0
11	туман	0	1	0	0	1	45	jintao	1	2	0	0	0
12	накрыл	0	1	0	0	1	46	,	0	4	0	0	1
13	города	0	1	0	0	0	47	hu	1	2	0	0	1
14	,	0	4	0	0	1	48	jintao	1	2	0	0	1
15	села	0	1	0	0	0	49	is	0	2	0	0	1
16	,	0	4	0	0	1	50	the	0	2	0	0	1
17	деревни	0	1	0	0	1	51	president	1	2	0	0	1
18	и	0	1	0	0	1	52	of	0	2	0	0	1
19	т	0	1	0	0	0	53	the	0	2	0	0	1
20	.	0	4	0	1	0	54	people's	1	2	0	0	1
21	п	0	1	0	0	0	55	republic	1	2	0	0	1
22	.	0	4	0	1	1	56	of	0	2	0	0	1
23	мерзко	1	1	0	0	1	57	china	1	2	0	0	0
24	и	0	1	0	0	1	58	.	0	4	1	1	1
25	зябко	0	1	0	0	0	59	his	1	2	0	0	1
26	!	0	4	1	0	1	60	email	0	2	0	0	1
27	according	1	2	0	0	1	61	is	0	2	0	0	1
28	to	0	2	0	0	1	62	hujin	0	2	0	0	0
29	the	0	2	0	0	1	63	@	0	0	0	0	0
30	website	0	2	0	0	1	64	mail	0	2	0	0	0
31	http	0	2	0	0	0	65	.	0	4	0	1	0
32	:	0	4	0	0	0	66	ch	0	2	0	0	0
33	/	0	5	0	0	0	67	.	0	4	1	1	1
34	/	0	5	0	0	0							

приведенным выше обозначениям для английского языка (отличается от них отсутствием концовки "_en"). К числу таковых не принадлежат только: _Accomplished — совершенный вид глагола, _VerbIntransitive — непереходный глагол.

Перечень обобщенных базовых элементов, получаемых в результате графематического анализа текста, приведен в табл. 10.

Исходный текст система разбивает на предложения следующим образом.

Сегодня 25-05-2010 в 16:00 туман накрыл города, села, деревни и т. п.

Мерзко и зябко!

According to the site http://ru.wikipedia.org/wiki/Hu_Jintao, Hu Jintao is the President of the People's Republic of China.

His email is hujin@mail.ch.

Таблица 9

Базовые элементы текста

№ элементов	Слово	Лемма	Морфологическая информация
1	сегодня	сегодня	_AdVerb
7	в	в	_Preposition_en
11	туман	туман	_Noun | _Singular | _Masculine | _Nominative | _NotAnimate
		туман	_Noun | _Singular | _Masculine | _Accusative | _NotAnimate
12	накрыл	накрыть	_Verb | _Past | _Singular | _Masculine | _Accomplished
13	города	город	_Noun | _Plural | _Masculine | _Nominative | _NotAnimate
		город	_Noun | _Plural | _Masculine | _Accusative | _NotAnimate
		город	_Noun | _Singular | _Masculine | _Genitive | _NotAnimate
15	села	село	_Noun | _Plural | _Neuter | _Nominative | _NotAnimate
		село	_Noun | _Singular | _Neuter | _Genitive | _NotAnimate
		село	_Noun | _Plural | _Neuter | _Accusative | _NotAnimate
		сесть	_Verb |_Past | _Singular | _Feminine | _Accomplished | _VerbIntransitive
17	деревни	деревня	_Noun | _Plural | _Feminine | _Nominative | _NotAnimate
		деревня	_Noun | _Singular | _Feminine | _Genitive | _NotAnimate
		деревня	_Noun | _Plural | _Feminine | _Accusative | _NotAnimate
18 24	и	и	_Interjection
		и	_Particle
		и	_Conjunction
23	мерзко	мерзкий	_Adjective | _Singular | _Neuter | _ShortFormOfAdj
		мерзко	_AdVerb
25	зябко	зябкий	_Adjective | _Singular | _Neuter | _ShortFormOfAdj
		зябко	_AdVerb
27	according	according	_Adjective_en
		accord	_Verb_en | 4st_Verb_form_en
28	to	to	_AdVerb_en
		to	_Preposition_en
		to	_Participle_en
29 50 53	the	the	_Article_en | _DefiniteArt_en
		the	_AdVerb_en
30	site	site	_Noun_en | _Singular_en
49 61	is	be	_Verb_en | _Verb_en n | _Singular_en
51	president	president	_Noun_en | _Singular_en
52 56	of	of	_Preposition_en
54	people's	people	_Noun_en | _Singular_en | _Prityag_en
55	republic	republic	_Noun_en | _Singular_en
57	china	china	_Noun_en | _Singular_en
59	his	he	_Pronoun_en | _Singular_en | _FaceThi_en | _Prityag_en
60	email	email	_Noun_en | _Singular_en
		email	_Verb_en

Таблица 10

Обобщенные базовые элементы текста

№ элемента	Обобщенный базовый элемент	SegmentType	WBegin	Wchief	Wend	№ элемента	Обобщенный базовый элемент	SegmentType	WBegin	Wchief	Wend
1	сегодня	Text	0	0	0	4	16:00	DataTime	7	0	9
2	25-05-2010	DataTime	1	0	5	5	туман	Text	10	10	10
3	в	Text	6	6	6	6	накрыл	Text	11	11	11

Продолжение табл. 10

№ элемента	Обобщенный базовый элемент	SegmentType	WBegin	Wchief	Wend	№ элемента	Обобщенный базовый элемент	SegmentType	WBegin	Wchief	Wend
7	города	Text	12	12	12	24	hu	Text	46	46	46
8	,		13	13	13	25	jintao	Text	47	47	47
9	села	Text	14	14	14	26	is	Text	48	48	48
10	,		15	15	15	27	the	Text	49	49	49
11	деревни	Text	16	16	16	28	president	Text	50	50	50
12	и	Text	17	17	17	29	of	Text	51	51	51
13	т.п.	Reduction	18	0	21	30	the	Text	52	52	52
14	мерзко	Text	22	22	22	31	people's	Text	53	53	53
15	и	Text	23	23	23	32	republic	Text	54	54	54
16	зябко	Text	24	24	24	33	of	Text	55	55	55
17	!		25	25	25	34	china	Text	56	0	56
18	according	Text	26	26	26	35	.		57	57	57
19	to	Text	27	27	27	36	his	Text	58	58	58
20	the	Text	28	28	28	37	email	Text	59	59	59
21	website	Text	29	29	29	38	is	Text	60	60	60
22	http://ru.wikipedia.org/wiki/hu_jintao	URL	30	0	44	39	hujin@mail.ch	email	61	0	65
23	,		45	45	45	40	.		57	57	57

Заключение

В настоящей работе предложен подход к реализации предсинтаксического анализа русско- и англоязычных текстов, базирующийся на многоуровневом представлении текста естественного языка. Разработанный для этого программный анализатор является первым компонентом лингвистического процессора, осуществляющего полный анализ текстов. Он предоставляет на выходе данные о структурных базовых элементах текста в форме, удобной для обработки другими компонентами.

Отличием предсинтаксической обработки текста, предложенной в данной работе, от других известных алгоритмов является выполнение морфологической разметки до графематического анализа. Новизна предложенного подхода состоит в следующем.

- Предложено использовать многоуровневое представление текста, которое, с одной стороны, сохраняет все данные, полученные на более ранних этапах обработки, а с другой стороны, позволяет получить представление о структурных элементах текста: лексических (слова) и нелексических (сокращения, аббревиатуры, адреса, даты и т. п.) единицах текста. Такое представление позволяет упростить синтаксический анализ текста. Кроме того, оно дает возможность изменить интерпретацию отдельных структурных элементов текста или их перечень, если такая необходимость возникнет на более поздних этапах обработки текста (синтаксический, семантический, прагматический), что делает лингвистический процессор более гибким и надежным.
- Получил дальнейшее распространение метод декларативного морфологического анализа слов, состоящий в явном задании парадигмы слова как набора словоформ, каждая из которых представлена написанием и морфологической информацией. При этом впервые сделана попытка системного изложения значений грамматических категорий английского языка, которая может быть применена в рамках данного метода. Разработан модуль декларативного морфологического анализа слов английского языка.

Представленный алгоритм предсинтаксического анализа текста может использоваться при создании компонентов лингвистического процессора для широкого класса интеллектуальных систем с элементами автоматической обработки естественно-языковых текстов.

Статья написана в ходе выполнения работ по проекту "Исследование и разработка программного обеспечения понимания неструктурированной текстовой информации на русском и английском языках на базе создания методов компьютерного полного лингвистического анализа", грант Минобрнауки 2012-1.4-07-514-0018.

Список литературы

1. **Автоматическая** обработка текста. URL: http://www.aot.ru/technology.html

2. **Сокирко А. В.** Семантические словари в автоматической обработке текста (по материалам системы ДИАЛИНГ). Дисс. ... канд. техн. наук. МГПИИЯ. М., 2001. 108 с. URL: http://www.aot.ru/docs/graphan.html.

3. **Дорохина Г. В., Павлюкова А. П.** Модуль морфологического анализа слов русского языка // Искусственный интеллект. 2004. № 3. С. 636—642.

4. **Дорохина Г. В., Трунов В. Ю., Шилова Е. В.** Модуль морфологического анализа без словаря слов русского языка // Искусственный интеллект. 2010. № 2. С. 32—36.

5. **Пат. № 78806** Україна, МПК G 06 F 17/30, G 06 F 7/76, G 06 F 12/00. Пристрій для збереження і пошуку рядкових величин та спосіб збереження і пошуку рядкових величин / Дорохіна Галина Володимирівна; заявник і власник Інститут проблем штучного інтелекту. — № a200500327; заявл. 14.01.2005; опубл. 25.04.2007.

6. **Дорохина Г. В., Журавлев А. О., Бондаренко Е. А.** Исследование алгоритма морфологического анализа слов с дефисным написанием // Системы и средства искусственного интеллекта. ССИИ—2012: материалы международной научной молодежной школы. Пос. Кацивели, АР Крым, Украина, 1—5 октября 2012. Донецк: ИПИИ "Наука і освіта", 2012. С. 17—24.

CONTENTS

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

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

Сдано в набор 05.08.2013 г. Подписано в печать 26.09.2013 г. Формат 60×88 1/8. Заказ PI1013
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.