ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

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

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/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 8
N 3
2017

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

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

УДК 519.682.2 DOI: 10.17587/prin.8.99-111

В. И. Шелехов, канд. техн. наук, зав. лаб., e-mail: vshel@iis.nsk.su,
Э. Г. Тумуров, мл. науч. сотр., e-mail: erdemus@gmail.com,
Институт систем информатики им. А. П. Ершова, г. Новосибирск

Технология автоматного программирования на примере программы управления лифтом

Технология автоматного программирования ориентирована на разработку простых, надежных и эффективных программ для класса реактивных систем. В качестве языка автоматного программирования используется формальный язык спецификации функциональных требований. Технология представлена в виде свода правил, определяющих правильный баланс в интеграции автоматного, предикатного и объектно-ориентированного программирования. Технология проиллюстрирована на примере программы управления лифтом.

Ключевые слова: *понимание программ, автоматное программирование, реактивная система, инженерия требований, спецификация требований*

Введение

Автоматное программирование [1—3] ориентировано на класс *реактивных систем*, реализующих взаимодействие с внешним окружением программы и реагирующих на определенный набор событий (сообщений) в окружении программы.

Определение требований — **первый этап** в разработке реактивной системы. *Требования* — совокупность утверждений о свойствах разрабатываемой программы. *Функциональные требования* определяют поведение программы. Их наиболее популярной формой являются *сценарии использования* (*use case*) [4]. Определение требований должно проводиться методами *инженерии требований* в соответствии со стандартом [5]. Для сложных производственных систем построение требований требует высокой квалификации специалистов в области инженерии требований.

Второй этап — формализация функциональных требований, содержательно сформулированных на первом этапе, в виде автоматной программы. При этом реализуется верификация формальных требований относительно содержательных требований, что обычно позволяет обнаружить значительное число ошибок.

Спецификация функциональных требований на формальном языке спецификаций не отличается принципиально от автоматной программы на некотором императивном языке. Отметим, что в мировой практике формальные языки спецификации требований используют лишь в 5 % случаев [6]. Фактически, построение автоматной программы является процессом формализации содержательных требований. Эта особенность является причиной многочисленных спекуляций по поводу программирования без программистов. В действительности, здесь нужны программисты, владеющие инженерией требований и навыками формализации требований.

Ввиду трудности формализации требований является существенным, в какой степени используемый язык автоматного программирования способствует хорошему пониманию автоматных программ. Другой важный аспект — применяемая технология автоматного программирования.

Язык автоматного программирования [1, 3] — компактный язык с синтаксисом в стиле C и C++, который строится как расширение *базисного языка* предикатного или императивного программирования. Транслятор с языка автоматного программирования преобразует автоматную структуру программы в конструкции базисного языка. В дополнение к операторному языку автоматного программирования имеется эквивалентный формальный *язык требований* с более компактной формой их записи, которая по структуре ближе к содержательным функциональным требованиям.

В настоящей статье описана развиваемая авторами технология автоматного программирования, эффективность использования которой иллюстрируется на примере программы управления лифтом. В разд. 1 дано общее описание базисного языка, которым является язык предикатного программирования Р [7]. Базис автоматного программирования определен в разд. 2. Язык требований описан в разд. 3. Технология автоматного программирования сформулирована в разд. 4; определены особенности интеграции автоматного, предикатного и объектно-ориентированного программирования. Процесс построения программы управления лифтом описан в разд. 5. Обзор работ представлен в разд. 6. Особенности верификации автоматных программ изложены в разд. 7.

1. Предикатное программирование

Предикатная программа относится к классу программ-функций [8] и является предикатом в форме вычислимого оператора. Язык предикатного программирования P [7] обладает большей выразительностью в сравнении с языками функционального программирования и по стилю он ближе к императивному программированию.

Полная предикатная программа состоит из набора рекурсивных *предикатных программ* на языке P следующего вида:

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

Необязательные конструкции предусловия и постусловия являются формулами на языке исчисления предикатов; они используются для улучшения понимания программ и для дедуктивной верификации [9—11]. Ниже представлены основные конструкции языка P: оператор присваивания, блок (оператор суперпозиции), условный оператор, вызов программы и описание переменных, используемое для аргументов, результатов и локальных переменных.

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

В предикатном программировании запрещены такие языковые конструкции, как циклы и указатели, значительно усложняющие программу. Вместо циклов используются рекурсивные программы, а вместо массивов и указателей — объекты алгебраических типов — списки и деревья.

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

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

Гиперфункции. Рассмотрим предикатную программу следующего вида:

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

Здесь x, y и z — непересекающиеся возможно пустые наборы переменных; P(x), C(x), S(x, y) и R(x, z) — логические утверждения, c — логическая переменная. Предположим, что все присваивания вида c = **true** и c = **false** — последние исполняемые операторы в теле предиката. Программа A может быть заменена следующей программой в виде *гиперфункции*:

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

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

Гиперфункция A имеет две *ветви* результатов: первая ветвь включает набор переменных y, вторая ветвь — набор z. *Метки* 1 и 2 — дополнительные параметры, определяющие два различных *выхода* гиперфункции. *Спецификация гиперфункции* состоит из двух частей. Утверждение после **pre** 1 есть предусловие первой ветви; предусловие второй ветви — отрицание предусловия первой ветви. Утверждения после **post** 1 и **post** 2 есть постусловия для первой и второй ветвей соответственно.

Аппарат гиперфункций является более общим и гибким по сравнению с известным механизмом обработки исключений, например, в таких языках, как Java и C++. Использование гиперфункций делает программу короче, быстрее и проще для понимания [11, 12]. Отметим, что гиперграфовая структура автоматной программы является естественным продолжением аппарата гиперфункций.

2. Язык автоматного программирования

Язык автоматного программирования строится расширением языка P [7]. *Автоматная программа* определяется следующей конструкцией:

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

Автоматная программа определяет конечный автомат. Вершина автомата — *управляющее состояние* программы. Ориентированная гипердуга автомата соответствует некоторому *сегменту кода* и связывает одну вершину с одной или несколькими другими вершинами.

Состояние автоматной программы определяется значениями набора переменных, модифицируемых в программе, за исключением локальных переменных. Взаимодействие с *внешним окружением* автоматной программы реализуется через прием и посылку *сообщений*, а также через *разделяемые переменные*, доступные в данной программе и в других программах из окружения данной программы.

Произвольный <сегмент кода> представляется конструкцией:

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

Исполнение <оператора> завершается либо оператором перехода вида #M, либо нормально; в последнем случае исполнение продолжится с начала следующего сегмента. Оператор #M, где M — имя управляющего состояния, реализует переход на начало сегмента, ассоциированного с управляющим состоянием M.

<Инвариант сегмента> должен быть истинным перед выполнением <оператора>, когда исполняемая программа приходит в данное управляющее состояние. Инвариант повышает понимание программы, его можно также использовать для верификации.

В качестве примера автоматной программы рассмотрим модуль операционной системы, реализующий следующий сценарий работы с пользователем (см. рисунок).

В управляющем состоянии login операционная система запрашивает имя пользователя. Если полученное от пользователя имя существует в системе, она переходит в управляющее состояние passw, иначе возвращается в состояние login. В состоянии passw система запрашивает пароль. Если поданная пользователем строка соответствует правильному паролю, то пользователь допускается к работе и система переходит в состояние session. При завершении работы пользователя система переходит в состояние login. Отметим, что реальная программа взаимодействия ОС с пользователем существенно сложнее; в частности, функция get _ string должна поставлять текст в зашифрованном виде.

В общем случае реактивная система определяется в виде композиции нескольких независимых автоматных программ, исполняемых параллельно и взаимодействующих между собой через сообщения и разделяемые переменные.

Неблокированный прием сообщения реализуется конструкцией <имя сообщения>(<параметры>). Ее значение — **true**, если из окружения получено сообщение с указанным именем.

Тип **time** используется для переменных и констант, значениями которых являются показания времени. Оператор **set** t, эквивалентный оператору **time** t = 0, реализует установку таймера переменной t. Ее изменение проводится непрерывно некоторым механизмом, не зависящим от автоматной программы.

3. Язык требований

Требования — совокупность утверждений относительно свойств разрабатываемой программы. *Функциональные требования* определяют поведение программы. Их наиболее популярной формой являются *сценарии использования* (*use case*) [4]. В нашем подходе они формализованы в виде правил на языке продукций [13], который обычно применяется для систем искусственного интеллекта. Это простой язык с высокой степенью декларативности. Спецификация на этом языке компактна и легко транслируется в автоматную программу, что позволяет использовать его как язык автоматного программирования.

Требование определяет один из вариантов функционирования автоматной программы и имеет следующую структуру:

$$\text{<условие}_1\text{>, <условие}_2\text{>, ...,}$$
$$\text{<условие}_n\text{>} \rightarrow \text{<действие}_1\text{>, ..., <действие}_m\text{>;}$$

Условиями являются: управляющие состояния, получаемые сообщения, логические выражения. *Действиями* являются: простые операторы, вызовы программ, посылаемые сообщения и итоговые управляющие состояния. Требование является спецификацией некоторого сегмента кода или его части. Семантика требования следующая: если в данный момент времени истинны все условия в левой части требования, то последовательно исполняется набор действий в правой части. Далее ограничимся детерминированными программами: для исполнения выбирается первое правило с истинными условиями.

Управляющее состояние в качестве <условия> означает утверждение того, что исполнение программы находится в данном управляющем состоянии. Оно должно быть первым в списке условий, и может быть опущено лишь в случае, когда автоматная программа имеет единственное управляющее состояние. Управляющее состояние в качестве <действия> — это следую-

```
login:   if (exists(get_string()))
                #passw
         else  #login
passw:   if (valid(get_string()))
                #session
         else  #passw
session: сессия_пользователя();
         #login
```

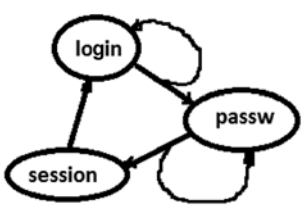

Схема работы ОС с пользователем: программа и ее автомат

щее управляющее состояние, с которого продолжится исполнение программы после завершения исполнения данного требования. Оно должно быть последним в списке действий.

4. Технология автоматного программирования

Разработка автоматной программы состоит из следующих этапов:

- определение требований в содержательной форме;
- формализация требований в виде набора правил;
- трансляция набора правил на язык автоматного программирования.

Фрагменты, соответствующие программам-функциям, разрабатываются в технологии предикатного программирования. Наконец, полная программа транслируется на язык C++.

На первом этапе определение требований фиксируется в виде *содержательного описания*. Рекомендуемая технология представлена в стандарте [5]. Формализация требований реализуется в следующей последовательности:

- специфицируются объекты внешнего окружения автоматной программы;
- описываются переменные *состояния* автоматной программы, определяются связи между переменными;
- фиксируются *управляющие состояния*; некоторые из них снабжаются инвариантами;
- определяется набор *требований* в виде системы правил. Каждое правило обеспечивает адекватную реакцию на определенное сообщение или событие.

Процесс формализации требований сопровождается их верификацией относительно содержательного описания.

В качестве примера рассмотрим формализацию требований для модуля, реализующего сценарий работы ОС с пользователем, представленный на рисунке.

Содержательное описание представлено в разд. 2.

Окружение:

```
message Get(string str); // получение строки
// от пользователя
```

Управляющие состояния: `login`, `passw`, `session`;

Требования:

```
login, Get(str) → User(str: #login: #passw);
passw, Get(str) → Password(str: #passw: #session);
session → UserSession(), login.
```

Здесь `User` и `Password` — гиперфункции с двумя ветвями без результирующих переменных.

Иногда (менее чем в 10 % случаев) непосредственная формализация требований дает неэффективную программу. В такой ситуации применяется метод трансформации требований [2], изменяющий автоматную структуру программы.

Интеграция с технологиями предикатного и императивного программирования. Автоматное программирование универсально. Любая программа-функция может быть запрограммирована в виде автоматной программы, которая, однако, будет значительно сложнее аналогичной предикатной или императивной программы, построенной обычными средствами. Поэтому *не следует использовать автоматное программирование для программ-функций*, являющихся фрагментами сегментов кода автоматных программ.

Необходима адекватная интеграция разных стилей программирования. *Не следует смешивать разные стили: части сегментов кода, соответствующие программам-функциям, должны быть представлены вызовами программ за исключением случаев, когда часть сегмента сводится к одному простому оператору.* Значительный по размеру фрагмент кода загромождает программу. Вынесение его из программы и оформление независимой подпрограммой улучшает понимание исходной программы. Отметим, что замена простых операторов вызовами в составе автоматной программы дает обратный эффект — избыточная структуризация введением дополнительного уровня иерархии усложняет программу.

Циклы в автоматной программе имеют другую природу по сравнению с циклами императивной программы. *Не рекомендуется использовать циклы типа* **`while`** *для конструирования автоматной программы.*

Интеграция с объектно-ориентированной технологией. Объектно-ориентированная декомпозиция позволяет существенно снизить сложность автоматной программы. *Внешнее окружение и состояние автоматной программы определяются как интерфейс класса в более простом и абстрактном виде.* При этом часть переменных состояния и связей между ними, а также детали внешнего окружения скрываются внутри класса.

Не следует прятать внутри класса автомат программы, т. е. управляющие состояния и сегменты кода, оставляя в интерфейсе лишь объекты внешнего окружения. Это худший способ реализации автоматной программы, выворачивающий ее наизнанку.

Баланс информационных и управляющих связей. Управляющие состояния можно заменить значениями дополнительной переменной в состоянии автоматной программы. И наоборот, переменную с ограниченным числом значений можно удалить из состояния программы с введением дополнительных управляющих состояний, эквивалентным образом заменяя информационные связи управляющими.

Модифицируем программу, представленную на рисунке, в стиле switch-технологии А. Шалыто [14]:

```
type STATE = enum(login, passw, session);
STATE state = login;
M: switch (state) {
     case login: if (exists(get _ string()))
                      state = passw
                 else state = login
     case passw: if (valid(get _ string()))
                      state = session
                 else state = passw
     case session: сессия_пользователя();
                   state = login
   }
#M
```

В модифицированной программе имеется лишь одно управляющее состояние M и единственный сегмент кода — оператор **switch**. Управляющие состояния login, passw и session становятся значениями переменной состояния state в модифицированной программе.

Исходная программа проще модифицированной. Детальное сравнение было проведено в работе [1, разд. 4, рис. 2]. Поэтому *не следует переводить управляющие состояния в состояние автоматной программы*. Однако замена информационных связей управляющими, приводящая к существенному увеличению программы, вряд ли оправдана.

Использование гиперфункций уменьшает число переменных состояния, тем самым упрощая автоматную программу. Отметим, что сложность автоматной программы зависит от числа переменных состояния программы, иногда экспоненциально.

Эргономическое правило. *Любая подпрограмма должна помещаться в пределах экрана монитора* — одновременная видимость всех ее информационно-управляющих связей упрощает ее восприятие и анализ. Применение гиперграфовой декомпозиции дает возможность удобно и гибко разделить программу на части произвольного размера.

Инварианты автоматной программы, ассоциированные с управляющими состояниями, принципиально отличаются от инвариантов циклов и инвариантов классов императивной программы. В большинстве случаев управляющее состояние не содержит инварианта. Иначе говоря, инвариант тождественно истинен.

5. Программа управления лифтом

Содержательное описание. *Лифт* установлен в здании с несколькими *этажами*. Этажи пронумерованы. Лифт либо *стоит* на одном из этажей с *открытой* или *закрытой* дверью, либо находится между этажами и *движется вверх* или *вниз*.

На каждом этаже есть две *кнопки* вызова лифта: одна — для движения *вверх*, другая — для движения *вниз*. На *нижнем* этаже нет кнопки для движения вниз, а на *верхнем* этаже — для движения вверх.

Внутри *кабины лифта* есть кнопки с номерами этажей. *Нажатие* одной из этих кнопок определяет команду остановки по прибытии лифта на соответствующий этаж.

Нажатые кнопки на этажах и в кабине лифта определяют текущее множество *заявок* на обслуживание пассажиров лифта. В момент завершения выполнения заявки соответствующая кнопка отжимается.

R1: Лифт движется в одном из направлений до тех пор, пока существуют заявки, реализуемые в этом направлении; когда заявки заканчиваются, направление движения лифта может быть изменено. Здесь R1 обозначает имя, введенное для идентификации приведенного требования. При отсутствии заявок в обоих направлениях лифт *останавливается* на текущем этаже.

По прибытии на этаж лифт либо останавливается на этаже, либо проходит мимо без остановки. Остановка реализуется при наличии заявки по данному этажу, т. е. при нажатой кнопке в кабине лифта, либо при нажатой кнопке на этаже, но не в направлении, противоположном движению лифта.

Решение об остановке на этаже принимается заранее вблизи этажа на определенном расстоянии по специальным датчикам. Если принято решение об остановке, включается торможение и лифт останавливается.

В случае остановки на этаже дверь лифта *открывается*. *Закрытие двери лифта* происходит через промежуток времени Tdoor, либо при нажатии кнопки "закрыть дверь" в кабине лифта. Если обнаружены помехи при закрытии дверей, они повторно открываются.

Окружение. Класс Лифт определяет набор методов — примитивов, используемых в программе Lift управления лифтом.

```
class Лифт {
     decisionIdle( : #idle : #start : #open);
     decisionClosed( : #idle : #start);
      starting(); // лифт начинает движение из
// состояния покоя вверх или вниз
       stopping(); // вблизи этажа включается
// торможение для остановки на этаже
     check _ floor(: #move: #stop); // вблизи этажа
// решается, остановиться или проехать мимо
     bool near _ floor(); // = true, когда движущийся
// лифт оказывается вблизи очередного этажа
      openDoor(); // реализуется открытие дверей
// лифта
     closeDoor(); // запускается процесс закрытия
// дверей лифта
      bool closeButton(); // = true при нажатии
// кнопки "закрыть дверь"
     bool closedDoor(); // = true, если закрытие
// дверей лифта завершено
     bool blockedDoor(); // = true, если закрытие
// дверей лифта блокировано пассажиром
            // поля и методы скрытой части класса:
     type DIR = enum (up, down, neutral); // тип
// состояния движения лифта
   DIR dir; // состояние движения лифта
      type FLOOR = first _ floor.. last _ floor;
// тип номера этажа
     FLOOR floor; // номер этажа, на котором стоит
// лифт или к которому подъезжает
   ...
}
```

Состояние dir = neutral устанавливается при отсутствии заявок после остановки лифта.

Для лифта, стоящего с закрытой дверью на некотором этаже, гиперфункция decisionIdle в зависимости от состояния кнопок определяет один из трех вариантов дальнейших действий: idle — оставаться в состоянии покоя; start — начать движение;

open — открыть дверь. После закрытия дверей лифта, остановившегося на некотором этаже, гиперфункция decisionClosed в зависимости от состояния кнопок выбирает один из выходов: idle — перейти в состояние покоя, start — начать движение.

Программа Lift управления лифтом использует только первые 11 методов класса Лифт. Остальная часть класса скрыта. Однако переменные скрытой части класса могут быть использованы в инвариантах. Отметим, что в скрытой части находится бóльшая часть окружения программы, в частности, весь механизм работы с кнопками и их световой индикацией.

Состояние. Объект класса Лифт.

Управляющие состояния программы Lift:

```
idle: inv dir = neutral; // лифт стоит на
// некотором этаже, двери закрыты
start: inv dir ≠ neutral; // лифт начинает
// движение в направлении dir
open; // перед открытием дверей лифт встал или
// уже стоит на некотором этаже
process Lift {
   idle → decisionIdle( : #idle : #start : #open);
   start → Movement( : #open);
   open → atFloor( : #idle : #start)
}
```

В управляющем состоянии idle лифт стоит. В соответствии с гиперфункцией decisionIdle произойдет переход снова в idle, пока не будет нажата кнопка на одном из этажей. Разумеется, кнопку может нажать и пассажир, задумавшийся в кабине лифта. Если нажата кнопка на том же этаже, на котором стоит лифт, происходит переход в управляющее состояние open. Гиперфункция decisionIdle также формирует новое значение переменной dir. Вызовы Movement и atFloor соответствуют процессам, которые определены ниже.

Управляющие состояния программы Movement:

```
start: inv dir ≠ neutral; // лифт начинает
// движение в направлении dir
move: inv dir ≠ neutral; // лифт движется
// в направлении dir
stop: inv dir ≠ neutral; // перед торможением
// лифт движется вблизи некоторого этажа
```

```
process Movement( : #open) {
   start → starting(), move;
  move, near _ floor() → check _ floor( : #move : #stop);
  stop → stopping(), open;
}
```

В процессе Movement метод starting() инициирует начало движения лифта с переходом в управляющее состояние move, в котором метод near _ floor() проверяет приближение движущегося лифта к очередному этажу. Когда лифт находится вблизи этажа, запускается гиперфункция check _ floor, определяющая, нужно ли останавливаться на этаже. Процесс переходит в управляющее состояние stop, если остановка нужна. Метод stopping запускает торможение лифта, и процесс Movement завершается внешним выходом open.

Управляющие состояния программы atFloor:

```
open; // лифт стоит на некотором этаже с закрытыми
// или полуоткрытыми дверями
opened; // двери лифта открыты
close; // двери лифта закрываются
```

Состояние программы atFloor:

```
time t; // время, прошедшее после полного
// открытия дверей лифта на текущем этаже
```

```
process atFloor( : #idle : #start) {
   open → openDoor(), set t, opened;
   opened, closeButton() or t ⩾ Tdoor →
closeDoor(), close;
   close, closedDoor() → decisionClosed(: #idle
: #start);
   close, blockedDoor() → open;
}
```

Процесс atFloor начинает работу в управляющем состоянии open. Вызов метода openDoor реализует открытие дверей лифта. Далее устанавливается таймер t и происходит переход в управляющее состояние opened. При нажатии на кнопку "закрыть дверь" или через промежуток времени Tdoor метод closeDoor запускает процесс закрытия дверей с переходом в управляющее состояние close. В соответствии с третьим и четвертым правилами процесса atFloor ожидается одно из двух событий: полное закрытие дверей при истинности closedDoor или блокировка дверей при истинности blockedDoor(). В случае блокировки дверей лифта процесс переходит в состояние open, в котором двери повторно открываются. В случае полного закрытия дверей гиперфункция decisionClosed в зависимости от состояния кнопок определяет вариант дальнейшего поведения лифта: перейти в состояние покоя или начать движение. В зависимости от выбранного варианта гиперфункция завершается по одному из выходов idle или start. Поскольку оба выхода — внешние для процесса atFloor, процесс atFloor завершает свою работу с возвратом в процесс Lift.

Программа. Сначала построим программы трех процессов по их требованиям.

```
process Lift {
  idle: decisionIdle( : #idle : #start : #open);
  start: Movement( : #open)
  open: atFloor( : #idle : #start)
}
process Movement( : #open) {
  start: starting() #move;
  move: if (near _ floor()) check _ floor( : #move: #stop);
        #move
  stop: stopping() #open;
}
process atFloor( : #idle : #start) {
        time t; // время, прошедшее после полного
// открытия дверей лифта
```

```
   open:   openDoor(); set t; #opened
   opened: if (closeButton() or t ⩾ Tdoor) {
closeDoor() #close }
           #opened
   close: if (closedDoor()) decisionClosed( : #idle
: #start);
            if (blockedDoor()) #open;
           #close
}
```

Подставляя тела программ Movement и atFloor на место их вызовов, после упрощений получаем окончательную автоматную программу:

```
time t; // время, прошедшее после полного
// открытия дверей лифта
process Lift {
   idle:  decisionIdle( : #idle : #start : #open);
   start: starting();
   move:  if (near_floor()) check_floor
( : #move : #stop);
       #move
   stop:   stopping();
   open:   openDoor(); set t;
   opened: if (closeButton() or t ⩾ Tdoor) {
closeDoor() #close }
           #opened
   close:  if (closedDoor()) decisionClosed
( : #idle : #start);
           if (blockedDoor()) #open;
           #close
}
```

Реализация класса Лифт. Сначала определим три массива кнопок: Up — для движения вверх на этажах, Down — для движения вниз и Cab — в кабине лифта.

```
type BUTTONS = array (bool, FLOOR); // тип массива
// кнопок
BUTTONS Up, Down, Cab;
```

Факт нажатия кнопки идентифицируется значением **true** соответствующего элемента массива; значение **false** соответствует отжатой кнопке. С каждой кнопкой связана независимая программа, реагирующая на нажатие (press) и отжатие (release) кнопки пассажиром:

```
process Button(BUTTONS Buttons, int j)
   { Cycle: if (press) Buttons[j] = true elsif
(release) Buttons[j] = false; #Cycle }
```

Программа работает параллельно с программой Lift, и поэтому возможен конфликт по доступу к переменной Buttons[j]. При этом не произойдет нарушения работы лифта, и нет необходимости применять средства синхронизации.

При открытии двери лифта срабатывает следующая программа:

```
Release() { Up[floor] = false; Down[floor] = false;
            Cab[floor] = false; }
```

Вызов программы Release вставляется в программу примитива openDoor. Нажатие и отжатие кнопок должно сопровождаться соответствующим изменением их световой индикации. Здесь этих действий нет, но они должны быть вставлены в реальную программу.

Ниже представлена программа гиперфункции decisionIdle, которая в зависимости от состояния кнопок реализует выбор одного из трех управляющих состояний: idle, start или open. Аргументами гиперфункции decisionIdle являются переменные floor, dir, Up, Down и Cab; результатом — dir. Отметим, что эти переменные не указываются при описании decisionIdle в интерфейсе класса Лифт, так как принадлежат скрытой части класса.

```
   decisionIdle(: #idle : DIR dir' #start: #open)
      pre dir = neutral
      pre idle: ∀ FLOOR j. ¬Up[j] & ¬Down[j] &
¬Cab[j]
   pre open: Up[floor] ∨ Down[floor] ∨ Cab[floor]
   post start: ∃ FLOOR j. (Up[j] ∨ Down[j] ∨Cab[j]) &
                          (j > floor & dir' = up ∨
                          j < floor & dir' = down)
{ decisionIdle_from(first_floor : #idle : DIR
dir' #start : #open) }
```

Здесь dir = neutral — общее предусловие. Предусловие по ветви idle реализуется при всех отжатых кнопках, а по ветви open — если нажата одна из кнопок на этаже. Предусловие для ветви start определяется как дополнение по отношению к предусловиям для ветвей idle и open в рамках общего предусловия. Постусловие для ветви start определяет условие на значение итоговой переменной dir. Постусловия для ветвей idle и open отсутствуют, поскольку по этим ветвям нет результирующих переменных.

Гиперфункция decisionIdle_from является более общей программой. Она работает начиная с этажа j в предположении, что все кнопки для этажей меньших j отжаты.

```
   decisionIdle_from(FLOOR j : #idle : DIR dir
#start : #open)
   pre dir = neutral & ∀ FLOOR p<j. ¬Up[p] &
¬Down[p] & ¬Cab[p]
   { if (Up[j] or Down[j] or Cab[j]) {
     if (j = floor) #open;
     if (j < floor) { dir = down; #start };
     dir = up; #start
   }
   if (j = last_floor) #idle;
   decisionIdle_from(j + 1 : #idle : DIR dir
#start : #open)
   }
```

Предусловия и постусловия по ветвям те же, что и у гиперфункции decisionIdle.

Программа гиперфункции decisionClosed использует подпрограмму inFloors для проверки наличия нажатых кнопок для этажей с номерами от j до k:

```
  formula InFloors(int j, k) = ∃ p = j..k.
(Up[p] ∨ Down[p] ∨ Cab[p]);
  inFloors(int j, k: bool b)
    pre j > k ∨ j, k ∈ FLOOR
    post b = InFloors(j, k)
  { if (j > k) b = false
    else if (Up[j] or Down[j] or Cab[j]) b = true
    else inFloors(j + 1, k: b)
  };
```

Подпрограммы below и above проверяют наличие нажатых кнопок, соответственно, ниже и выше этажа floor:

```
  formula Below() = InFloors(first _ floor, floor-1);
  formula Above() = InFloors(floor + 1, last _ floor);
  below( : bool b) post b = Below() { inFloors(first _
floor, floor-1 : b) };
  above( : bool b) post b = Above() { inFloors(floor + 1,
last _ floor: b) };
```

Гиперфункция decisionClosed в зависимости от состояния кнопок определяет выбор между управляющими состояниями idle и start. Кроме того, определяется направление движения dir. Действует правило: лифт не может изменить направления движения, пока по ходу движения лифта имеются нажатые кнопки.

```
decisionClosed(DIR dir : DIR dir′ #idle : DIR dir′ #start)
  pre dir ≠ neutral
  pre idle: ∀ FLOOR j≠ floor. ¬Up[j] & ¬Down[j] &
¬Cab[j]
  post idle: dir′ = neutral
  post start: (Below() & dir = down ⇒ dir′ = down) &
              (Above() & dir = up ⇒ dir′ = up) &
          (Below() & dir′ = down ∨ Above() & dir′ = up);
{ if (dir = down & below() or dir = up & above()) #start
  else if (below()) { dir′ = down; #start }
  else if (above()) { dir′ = up; #start }
  else { dir′ = neutral; #idle }
}
```

Поскольку проверяются все этажи, кроме текущего, нажатие одной из кнопок для текущего этажа floor не препятствует переходу в состояние idle. Требуемое открывание дверей лифта реализуется после срабатывания decisionIdle в состоянии idle.

Примитив starting запускает движение лифта из состояния покоя вверх при dir = up или вниз при dir = down. В дополнение к этому в качестве текущего этажа соответственно назначается следующий по ходу движения.

```
  starting(FLOOR floor: FLOOR floor′)
    pre dir = up ∨ dir = down
  { if (dir = down) { floor′ = floor − 1; startingDown() }
    else { floor′ = floor + 1; startingUp() }
  };
```

Примитивы startingDown и startingUp реализуют запуск механизма движения лифта.

Гиперфункция check _ floor, запускаемая при приближении к очередному этажу, решает, остановиться или проехать мимо. Остановка реализуется при нажатой кнопке в кабине лифта, либо при нажатой кнопке на этаже, но не в направлени, противоположном движению лифта. Лифт останавливается также при отсутствии нажатых кнопок в направлении движения лифта. Если лифт проходит мимо, значение текущего этажа floor заменяется на следующий по ходу движения.

```
  check _ floor( : FLOOR floor #move : #stop)
    pre dir = up ∨ dir = down
    pre stop: Cab[floor] ∨ dir = down & (Down[floor] ∨ ¬Below()) ∨
                           dir = up & (Up[floor] ∨ ¬Above())
  { if (Cab[floor]) #stop;
    if (dir = down) { if (Down[floor] or not below()) #stop }
    else { if (Up[floor] or not above()) #stop };
    if (dir = down) floor′ = floor − 1 else floor′ = floor + 1;
    #move
  };
```

Реализация остальных примитивов класса Лифт использует датчики и механизмы, обеспечивающие функционирование лифта. Эти датчики и механизмы предварительно должны быть определены в классе Лифт. Реализация оставшихся примитивов не представляет особенных трудностей. Отметим лишь, что в реализации примитива closeDoor() дополнительно требуется запустить отжатие кнопки "закрыть дверь".

Моделирование управления лифтом. Для предикатных программ, являющихся примитивами класса Лифт, применяется оптимизирующая трансформация на императивное расширение языка P, с которого программа легко кодируется на язык C++. Лифт моделируется в виде графического интерфейса пользователя, созданного с помощью инструмента Qt. Модель управления лифта работает под ОС Windows и доступна по адресу: http://persons.iis.nsk.su/files/persons/pages/lift-win32.zip

Другие варианты. Возможна другая версия программы без переменной dir с увеличением числа управляющих состояний и раздваиванием фрагментов программы, реализующих движение. Вместо процесса Movement появятся процессы Movement _ up и Movement _ down. Аналогичным образом расклеиваются decisionClosed, check _ floor, starting и другие примитивы. Программа увеличивается по размеру почти вдвое, но при этом упрощается. Заметим, что упрощаются лишь примитивы, принадлежащие классу программ-функций. Удвоение числа управляющих состояний существенно усложняет автоматную программу. Поскольку критичной является сложность именно автоматной части программы, новая версия программы хуже исходной. Можно пойти дальше и устранить переменную floor. При этом следует зафиксировать число этажей,

например, пять, и реализовать примитивы перемещения лифта между соседними этажами. Полезность этой версии сомнительна.

Замечания. Представленная выше программа управления лифтом на порядок проще аналогичных программ, описанных в работах [15—19]. Простота программы обусловлена переносом ее информационных связей внутрь класса `Лифт`. Программа универсальна, поскольку вся специфика внешнего окружения (кнопок и их индикации, датчиков вблизи этажа, а также закрытия и блокировки дверей) находится в скрытой части класса `Лифт`. Ряд особенностей работы лифта, приведенных в содержательном описании, также оказалась в скрытой части.

Чтобы программа была корректной, необходимо гарантировать, что значения переменных `dir`, `floor`, `Up`, `Down` и `Cab` правильно отражают реальное состояние лифта. Например, в случае отказа одного из датчиков вблизи этажа не будет запущена программа `check _ floor`, вследствие чего значение текущего этажа `floor` не будет соответствующим образом изменено; дальнейшая работа программы `Lift` при неправильном значении `floor` будет ошибочной. Чтобы повысить отказоустойчивость программы `Lift`, в качестве результата примитива `near _ floor()` следует выдавать не логическое значение, а номер текущего этажа.

6. Обзор работ

Современный пассажирский лифт [20] является сложным техническим сооружением. Программа управления работой лифта нетривиальна. Ее нередко используют для демонстрации технологии программирования. Примеры программ управления лифтом можно найти в работах [15—19]. В книге Д. Кнута представлена нетривиальная реализация в форме сопрограмм [18]. В книге Х. Гома даны различные виды диаграмм UML [15]. Автоматные методы программирования в сочетании с технологией объектно-ориентированного программирования представлены в работах [15, 16]. В руководстве [19] определяется серия из пяти последовательно уточняемых моделей лифта в технологии Event-B. Предложенная авторами программа управления лифтом существенно проще и короче программ в упомянутых выше работах. Есть три составляющие, обеспечивающие простоту нашей программы `Lift`: гиперграфовая композиция в сочетании с механизмом гиперфункций, объектная ориентированность и реализация примитивов вне автоматной программы на языке P.

Формальный метод моделирования и анализа реактивных систем Event-B [21] определяет процесс построения серии моделей исходя из содержательного описания требований к реактивной системе. Каждая очередная модель является детализацией (*refinement*) предыдущей модели. Корректность моделей и согласованность моделей различных уровней обеспечивается проведением математических доказательств истинности инвариантов. Следует отметить неадекватность терминологии. *Событие* (*event*) в Event-B включает действия, реализующие реакцию на событие, т. е. сопоставимо с требованием в предлагаемом нами подходе. По мнению авторов, нет никаких оснований называть аксиомами ограничения на значения входных переменных модели.

Система управления лифтом рассматривается в руководстве [19] в качестве примера использования технологии Event-B. Определяется пять уровней:

1) лифт без дверей и кнопок;
2) добавление дверей лифта;
3) добавление дверей на этажах;
4) добавление кнопок в лифте;
5) добавление кнопок на этажах, по одной кнопке на этаж.

На пятом уровне используется 17 событий и 8 переменных: номер текущего этажа, статус лифта (*moving*, *stopped*, *idle*), направление движения (*up*, *down*), статус дверей лифта (*closed*, *opening*, *open*, *closing*), статус дверей на этажах, массив кнопок лифта, подмножество нажатых кнопок, массив кнопок на этажах. В нашей программе используется десять требований, две переменных (`dir` и `floor`) и три массива кнопок. Определяя более сложную функциональность, наша программа существенно проще. Переменные, определяющие статусы лифта и дверей, не нужны, поскольку все действия с дверями локализованы в подпрограмме `atFloor`, работающей при остановленном лифте и завершающей работу с гарантией закрытия дверей. Подпрограмма `atFloor` появилась в результате гиперграфовой декомпозиции программы `Lift`, что обеспечивает ее простоту.

Уже модель первого уровня (без дверей и кнопок), описанная в работе [19], оказалась значительно сложнее нашей реализации. Полная коллекция из пяти уровней является громоздкой. Технология Event-B [21] декларирует возможность экстракции исполняемой программы из коллекции моделей. Даже если это возможно, польза от такой программы сомнительна. Отметим, что в нашем подходе автоматная программа может быть оттранслирована в эффективную программу на язык C++.

Язык интервальной темпоральной логики, разработанный Джоном Рушби для верификации систем реального времени, демонстрировался для спецификации системы управления лифтом [22]. Спецификация в виде 31 темпоральной формулы достаточно сложна; она намного сложнее представленной в данной работе. Работа [22] по интервальной логике не имела продолжений[1], поскольку появились более предпочтительные подходы к спецификации, например, диаграммы *use case*.

Разрабатываемая технология предикатного программирования до недавнего времени не имела в мире аналогов. Впервые появился язык, похожий

[1] Одно исключение: работа [22] использовалась А. А. Калентьевым и А. А. Тюгашевым при построении собственного языка интервальной логики для спецификации программ реального времени.

на язык предикатного программирования P. Функциональный язык доказательного программирования Smart разработан французской компанией Prove&Run[2]. Корректность программ, а также отсутствие уязвимостей обеспечиваются дедуктивной верификацией. Язык Smart содержит конструкции, аналогичные гиперфункциям. Программа может быть оттранслирована на языки C и Java. Полное описание языка недоступно. Некоторые особенности языка Smart изложены в работах [23, 24].

В событийно-ориентированной парадигме [25, 26] программа представлена в виде цикла, в котором последовательно просматривается определенный набор событий (сообщений). Для всякого события запускается соответствующий обработчик события. Таким образом, программа составляется из набора независимых обработчиков. В частности, графический интерфейс пользователя (GUI) во всех реализующих его инструментах определяется именно в такой архитектуре. Авторы данной парадигмы признают, что программирование в ней является сложным. Причина сложности в том, что в обработчике события неестественным образом перемешиваются части программ, которые в автоматной программе принадлежат сегментам кода для разных управляющих состояний. Событийно-ориентированный стиль программирования иногда применяется во избежание параллелизма в программе, нередко используя при этом сопрограммную схему взаимодействия. Отметим, что замена параллельного исполнения сопрограммным является преобразованием, существенно усложняющим программу, см., например, работу [27]. Событийно-ориентированное программирование должно быть ограничено первичной обработкой событий (сообщений) для последующей передачи их автоматной программе, исполняемой параллельно. В частности, графические интерфейсы пользователя должны быть ориентированы лишь для построения разнообразного гибкого внешнего окружения автоматной программы, но не ее самой.

Обзор работ по автоматному программированию представлен также в статьях [1—3].

7. Верификация автоматных программ

Валидация требований, по которым строится автоматная программа, заключается в их проверке на соответствие потребностям пассажиров лифта. Обычно здесь применяется моделирование. Результаты валидации оцениваются совместно разработчиком и заказчиком. Например, может оцениваться правильность решения о двух кнопках на каждом этаже вместо одной. Требование `R1` также может быть предметом оценки и сопоставления с другими возможными стратегиями выполнения заявок.

Верификация автоматной программы реализуется относительно ее спецификации, в роли которой выступают инварианты управляющих состояний. Спецификацией являются также и требования. Однако, поскольку программа транслируется из требований, то здесь нет предмета для верификации.

Формальная верификация автоматных программ может быть реализована следующим образом. Автоматная программа преобразуется в эквивалентный набор логических формул в соответствии с формальной операционной семантикой предикатных программ [28]. Формула для процесса, например, `Movement`, строится в виде конъюнкции формул для требований, составляющих процесс. Вызов предикатной программы заменяется ее спецификацией, т. е. конъюнкцией предусловия и постусловия. Для каждого сегмента кода (или требования) в качестве предусловия выступает инвариант исходного управляющего состояния для данного сегмента, а в качестве постусловия — инвариант следующего управляющего состояния. Если инварианты отсутствуют, верификация ограничивается проверкой истинности предусловий для всех используемых вызовов и операций. Например, для вызова примитива `starting` проверяется истинность предусловия $\texttt{dir} = \texttt{up} \vee \texttt{dir} = \texttt{down}$. Дедуктивная верификация используемых предикатных программ реализуется методами, описанными в работах [9—11].

Инварианты управляющих состояний, если присутствуют, являются довольно слабыми, как, например, инвариант $\texttt{dir} \neq \texttt{neutral}$. По этой причине верификация автоматной программы оказывается принципиально неполной. Проведение верификации не гарантирует отсутствия ошибок в автоматной программе. Не помогут также и дополнительные инварианты внутри сегментов кода. Факт неполноты верификации реактивных систем серьезно недооценивается во многих работах.

Модель в виде предусловия и постусловия для сегментов кода неадекватна, поскольку сегмент кода состоит из нескольких разнородных кусков и не представляет цельной программы [3].

Объектами верификации могут быть также *свойства* автоматной программы, обычно формулируемые на языке темпоральной логики. Свойства программы управления лифтом определяются исходя из основного назначения лифта — перевозить пассажиров лифта с одного этажа на другой. Главное свойство: любая заявка, инициируемая нажатием кнопки, должна быть обслужена, разумеется, при условии, что кнопка не будет отжата. Точнее, это свойство формулируется следующим образом. Если для некоторого этажа `j` нажата одна из кнопок на этаже или соответствующая кнопка в кабине лифта, то через определенное время лифт гарантированно подойдет к этажу `j` и откроет двери. Формально данное условие записывается формулой

$$\Box\ \forall\ \texttt{FLOOR j.}\ (\texttt{Up[j]} \vee \texttt{Down[j]} \vee \texttt{Cab[j]})\ \&\ (\Box\ \neg\texttt{release}) \Rightarrow \Diamond\ \texttt{floor} = \texttt{j}\ \&\ \texttt{open}$$

Здесь "$\Box$" и "$\Diamond$" - темпоральные кванторы. Формула вида $\Box$A определяет следующее утверждение:

[2] www.provenrun.com

в любой момент работы лифта будет истинной формула A. Формула вида ◊A есть утверждение: через конечный промежуток времени работы лифта в будущем будет истинна формула A.

Свойства программы на языке темпоральной логики могут быть верифицированы с помощью инструментов проверки на модели (*model checking*). При этом автоматная программа должна быть оттранслирована в эквивалентный набор логических формул в соответствии с формальной операционной семантикой [28].

Заключение

Программа управления работой лифта нетривиальна. Являясь автоматной программой, она содержит фрагменты, относящиеся к классу программ-функций, для построения которых нужно применять другие методы, отличные от методов автоматного программирования. Другая особенность — сложность внешнего окружения. Для упрощения программы применяется объектно-ориентированный подход. Взаимодействие с окружением реализуется через интерфейс класса, а детали окружения спрятаны внутри класса.

Для понимания программы Lift наиболее подходящим является ее описание на языке требований в виде трех процессов Lift, Movement и atFloor. Построение программ примитивов, таких как decisionIdle и check _ floor, проводится в технологии предикатного программирования; автоматные программы таких примитивов оказались бы существенно сложнее соответствующих предикатных программ.

Представленная в настоящей статье программа управления лифтом существенно проще и короче программ, описанных в работах [15—19], демонстрирующих различные технологии. Есть три составляющие, обеспечивающие простоту программы Lift: гиперграфовая автоматная композиция в сочетании с механизмом гиперфункций, объектная ориентированность и реализация примитивов вне автоматной программы на языке P. Отметим, что в оценке сложности программы критичной является сложность автоматной части программы.

Работа выполнена при поддержке РФФИ, грант № 16-01-00498.

Список литературы

1. **Шелехов В. И.** Язык и технология автоматного программирования // Программная инженерия. 2014. № 4. С. 3—15.

2. **Шелехов В. И.** Оптимизация автоматных программ методом трансформации требований // Программная инженерия. 2015. № 11. С. 3—13.

3. **Шелехов В. И.** Разработка автоматных программ на базе определения требований // Системная информатика. 2014. № 4. С. 1—29.

4. **Cockburn A.** Writing Effective Use Cases. Addison-Wesley, 2001. 270 p.

5. **Systems** and software engineering — Life cycle processes — Requirements engineering. ISO/IEC/ IEEE 29148. URL: http://datateca.unad.edu.co/contenidos/204019/EConocimiento/M11-IEEE_29148-2011.pdf.

6. **Mich L., Franch M., Novi Inverardi P.** Market research for requirements analysis using linguistic tools // Requirements Engineering. 2004. N. 9 (1). P. 40—56.

7. **Карнаухов Н. С., Першин Д. Ю., Шелехов В. И.** Язык предикатного программирования P. Новосибирск, Препр. ИСИ СО РАН; № 153. 2010. 42 с.

8. **Шелехов В. И.** Классификация программ, ориентированная на технологию программирования // Программная инженерия. 2016. Том 7, № 12. С. 531—538.

9. **Shelekhov V. I.** Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements // Automatic Control and Computer Sciences. 2011. Vol. 45, N. 7. P. 421—427.

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

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

12. **Шелехов В. И.** Предикатное программирование. Учебное пособие. Новосибирск: НГУ, 2009. 109 с.

13. **Klahr D., Langley P., Neches R.** Production System Models of Learning and Development. — Cambridge, Mass.: The MIT Press, 1987. 467 p.

14. **Шалыто А. А.** SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

15. **Гома Х.** UML. Проектирование систем реального времени, параллельных и распределенных приложений. М.: ДМК Пресс, 2002. 684 с.

16. **Наумов А. С., Шалыто А. А.** Система управления лифтом. Санкт-Петербург, 2003. 51 с. URL: http://is.ifmo.ru/download/elevator_a.pdf

17. **Решетников Е. О., Смачных М. В.** Система управления пассажирским лифтом. Санкт-Петербургский государственный университет информационных технологий, механики и оптики. 2006. URL: http://is.ifmo.ru/download/umlift.pdf

18. **Кнут Д. Э.** Искусство Программирования. Том 1. Основные Алгоритмы. М.: Вильямс, 2006. разд. 2.2.5.

19. **Robinson K.** System Modelling & Design. Using Event-B. Draft book. 2012. 142 p. URL: http://www.cse.unsw.edu.au/~cs2111/PDF/SMD-KAR.pdf

20. **Манухин С. Б., Нелидов И. К.** Устройство, техническое обслуживание и ремонт лифтов. М.:Академия, 2004. 336 с.

21. **Abrial J.-R.** Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010. 586 p.

22. **Rushby J.** Specifying real-time systems with interval logic (SuDoc NAS 1.26:181804). SRI International. Menlo Park, CA, USA, 1988. 81 p.

23. **Andreescu O. F., Jensen T., Lescuyer S.** Dependency Analysis of Functional Specifications with Algebraic Data Structures // ICFEM'15. LNCS 9407, 2015. P. 116—133.

24. **Lescuyer S.** Towards a verified isolation micro-kernel. MILS: Architecture and Assurance for Secure Systems, Amsterdam, 2015. 8 p.

25. **Vlissides J., Johnson R., Helm R., Gamma E.** Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 431 p.

26. **Ferg S.** Event-Driven Programming: Introduction, Tutorial, History. SourceForge, 2006. 59 p.

27. **Шелехов В. И., Демаков И. В.** Спецификация и реализация радиус-сервера интернет-телефонии // Методы предикатного программирования. 2006. Вып. 2. С. 40—63.

28. **Шелехов В. И.** Семантика языка предикатного программирования // Материалы конференции ЗОНТ-15. Новосибирск, 2015. URL: http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf

Applying Automata-based Software Engineering for the Lift Control Program

V. I. Shelekhov, vshel@iis.nsk.su, **E. G. Tumurov,** erdemus@gmail.com, A. P. Ershov Institute of Informatics Systems, Novosibirsk, 630090, Russian Federation

Corresponding author:

Shelekhov Vladimir I., Head of Laboratory, A. P. Ershov Institute of Informatics Systems, Novosibirsk, 630090, Russian Federation
E-mail: vshel@iis.nsk.su

Received on December 8, 2016
Accepted on December 15, 2016

Automata-based software engineering is intended to develop the simple, reliable, and effective programs from the class of reactive systems. A formal rule-based language is used for a specification of functional requirements of a reactive system. This specification language is easily translated to the automata-based programming language that extends the predicate programming language P. In this article, automata-based methods of software engineering are presented in the form of true balance between integrated automata-based, predicate, and object-oriented software engineering. These methods are applied for the lift control programs, which are often used for illustration of new software engineering methods.

The lift control program is not trivial. Being automata-based in general, it includes fragments that are not automata-based and should be programmed in other paradigms. To simplify the complex environment of the lift control program, only objects of the `Lift` *class interface are used in the main program. The implementation of the* `Lift` *class hides numerous details of the environment. The lift control program presented is the most easy and short among other six lift control programs demonstrated in the articles and books as illustrations of different successful methods. There are three factors why our lift control program is the best: hypergraph automata composition of the program with the usage of the hyperfunctions, hiding detail inside* `Lift`*, and development primitives outside of automata-based engineering.*

Keywords: *program comprehension, automata-based software engineering, reactive system, requirement engineering, requirement specification.*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project nos. 16-01-00498*

For citation:

Shelekhov V. I., Tumurov E. G. Applying Automata-based Software Engineering for the Lift Control Program, *Programmnaya Ingeneria*, 2017, vol. 8, no. 2, pp. 99—111.

DOI: 10.17587/prin.8.99-111

References

1. **Shelekhov V. I.** Jazyk i tehnologija avtomatnogo programmirovanija (Automata-Based Software Engineering: the Language and Development Methods), *Programmnaya Ingeneria*, 2014, no. 4, pp. 3—15 (in Russian).

2. **Shelekhov V. I.** Optimizacia avtomatnih program metodom transformacii trebovanii (Automata-based Program Optimization by Applying Requirement Transformations), *Programmnaya Ingeneria*, 2015, no. 11, pp. 3—13 (in Russian).

3. **Shelekhov V. I.** Razrabotka avtomatnykh programm na baze opredeleniya trebovaniy (Automata-based programming on the base of requirements specification), *Sistemnaya Informatika*, 2014, no. 4. pp. 1—29 (in Russian).

4. **Cockburn A**. Writing Effective Use Cases, *Addison-Wesley*, 2001, 270 p.

5. **Systems** and software engineering — Life cycle processes — Requirements engineering. ISO/IEC/ IEEE 29148, available at: http://datateca.unad.edu.co/contenidos/204019/EConocimiento/M11-IEEE_29148-2011.pdf

6. **Mich L., Franch M., Novi Inverardi P.** Market research for requirements analysis using linguistic tools, *Requirements Engineering,* 2004, no. 9 (1), pp. 40—56.

7. **Kharnaukhov N. S., Perchine D. Ju., Shelekhov V. I.** *Yazyk predikatnogo programmirovaniya P* (The predicate programming language P), Preprint no. 153, Novosibirsk, ISI SB RAN, 2010, 42 p. (in Russian).

8. **Shelekhov V. I.** Klassifikacija programm, orientirovannaja na tehnologiju programmirovanija (Program Classification in Software Engineering), *Programmnaya Ingeneria,* 2016, vol. 7, no. 12, pp. 531—538 (in Russian).

9. **Shelekhov V. I.** Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements, *Automatic Control and Computer Sciences*, 2011, vol. 45, no. 7, pp. 421—427.

10. **Shelekhov V. I.** Verifikatsiya i sintez effektivnykh programm standartnykh funktsiy v tekhnologii predikatnogo programmirovaniya **(**Verification and synthesis of effective programs for standard

functions in predicate software engineering), *Programmnaya Ingeneria*, 2011, no. 2, pp. 14—21 (in Russian).

11. **Shelekhov V. I.** *Razrabotka i verifikatsiya algoritmov piramidalnoy sortirovki v tekhnologii predikatnogo programmirovaniya* (Development and verification of the heapsort algorithms in predicate software engineering), Preprint no. 164. Novosibirsk, ISI SB RAN, 2012, 30 p. (in Russian).

12. **Shelekhov V. I.** *Predikatnoye programmirovaniye. Uchebnoye posobiye* (Predicate programming. Tutorial), Novosibirsk: NSU, 2009, 109 p. (in Russian).

13. **Klahr D., Langley P., Neches R.** *Production System Models of Learning and Development*, Cambridge, Mass., The MIT Press, 1987, 467 p.

14. **Shalyto A. A.** *SWITCH-tehnologija. Algoritmizacia i programmirovanie zadach logicheskovj upravlenia* (SWITCH-technology. Algorithmic and Programming Methods in Solution of Logic Control Problems), St. Petersburg, Nauka (Science), 1998, 628 p. (in Russian).

15. **Gomma H**. UML. *Proektirovanie system realnogo vremeni, parallelnih i raspredelennih prilozhenii* (Designing concurrent, distributed and real-time applications with UML), Moscow, DMK Press, 2002, 684 p. (in Russian).

16. **Naumov A. S., Shalyto A. A**. *Sistema upravlenia liftom* (Lift control system), St. Petersburg, 2003. 51 p., available at: http://is.ifmo.ru/download/elevator_a.pdf (in Russian).

17. **Reshetnikhov E. O., Smachnyh M. V**. *Systema upravlenia liftom* (Passenger lift control system), ITMO, St. Petersburg, 2006 (in Russian).

18. **Knuth D. E**. *Iskustvo programirovanija. Tom 1. Osnovnie algoritmy* (The Art of Computer Programming. Fundamental algorithms), Moscow, Williams, 2006 (in Russian).

19. **Robinson K**. *System Modelling & Design. Using Event-B. Draft book*, 2012, 142 p.

20. **Manuhin S. B., Nelidov I. K**. *Ustoistvo, tehnichesroe obsluzhivanie liftov* (Device, maintenance and repair of elevators), Moscow, Akademia, 2004, 336 p. (in Russian).

21. **Abrial J.-R**. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010, 586 p.

22. **Rushby J**. *Specifying real-time systems with interval logic* (SuDoc NAS 1.26:181804), SRI International, Menlo Park, CA, USA, 1988. 81 p.

23. **Andreescu O. F., Jensen T., Lescuyer S**. Dependency Analysis of Functional Specifications with Algebraic Data Structures, *ICFEM'15. LNCS 9407*, 2015, pp. 116—133.

24. **Lescuyer S.** *Towards a verified isolation micro-kernel. MILS: Architecture and Assurance for Secure Systems*, Amsterdam, 2015, 8 p.

25. **Vlissides J., Johnson R., Helm R., Gamma E**. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994, 431 p.

26. **Ferg S**. *Event-Driven Programming: Introduction, Tutorial, History*, SourceForge, 2006, 59 p.

27. **Shelekhov V. I.**, **Demakhov I. V**. Specifikacija i realizacija radius-servera internet-telefonii (Specification and implementation of Internet telephony radius server), *Metody predicatnogo programmirovanija*, 2006, no. 2, pp. 40—63 (in Russian).

28. **Shelekhov V. I.** Semantika jazyka predikatnogo programmirovanija (Formal Semantics of the P Predicate Language). *ZONT-15*. Novosibirsk, 2015, available at: http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf (in Russian).

ИНФОРМАЦИЯ

УДК 004.04 + 004.4 DOI: 10.17587/prin.8.112-119

В. М. Димитров, преподаватель, e-mail: dimitrov@cs.petrsu.ru, **А. В. Воронин**, д-р техн. наук, ректор, e-mail: voronin@petrsu.ru, **Ю. А. Богоявленский**, канд. техн. наук, зав. каф., e-mail: ybgv@cs.petrsu.ru, Петрозаводский государственный университет

Лаконичный язык запросов LaOQL на основе связей в объектной модели данных

Представлен лаконичный язык запросов, который позволяет сократить длину текста запроса и, соответственно, время на его подготовку. Для лаконизации предложено использовать информацию о связях между классами объектно-ориентированной модели данных, на которой основана объектно-ориентированная базы данных. Авторами был использован подход к разработке объектно-ориентированной базы данных, описанный в стандарте ODMG 3.0, а конкретно следующие элементы: объект; литерал (literal); тип объекта; состояние (атрибуты и отношения); привязка (binding) к языку программирования Java. Такие элементы, как идентификация, именование, создание, поведение, привязки к языкам C++ и Smalltalk не использовались.

Для лаконизации также предложены механизмы формирования сокращенных имен классов, исключения ключевых слов, использования значений атрибутов по умолчанию.

Ключевые слова: *объектно-ориентированные базы данных, язык запросов, вычислительные сети*

Введение

Преимущества и недостатки объектно-ориентированных баз данных (ООБД) изучают давно и интенсивно [1—3]. Для манипуляций с данными в ООБД применяют специальные языки запросов. При этом текст запроса имеет длину от сотен до тысяч символов. Однако существуют области применения ООБД, в которых требуется очень интенсивно формулировать и выполнять запросы. Например, если ООБД содержит данные о вычислительной сети, то интенсивная посылка запросов необходима при поиске источников аномалий трафика, исследовании конфигурации, при решении других подобных задач. В таких условиях лаконичность текста запроса позволяет существенно повысить производительность труда пользователя ООБД (в данном примере администратора или исследователя вычислительной сети).

В статье описан разработанный авторами язык LaOQL — лаконичный язык запросов к ООБД. Для лаконизации предлагается прежде всего сокращать в запросах длину выражений путей в объектном графе за счет использования информации о связях между классами модели данных ООБД, заданной диаграммой классов в нотации UML. Эта диаграмма по предлагаемым авторами правилам преобразуется в граф классов, на основе которого формируется набор путей (карта предметной области, КПО), которые будут использованы для автоматизации построения выражений путей в запросе на поиск в объектном графе.

Авторами выполнено сравнительное исследование длины текстов запросов на языках LaOQL и на одной из реализаций стандарта OQL [4] — языке Hibernate Query Language (HQL) [5]. Исследование показало, что благодаря предложенному подходу длина текста запроса сокращается на 30...90 %, при этом все возможности языка HQL сохраняются. Проведены также эксперименты по сравнению производительности (в качестве такой характеристики было выбрано время выполнения запроса) и потреблению памяти языков HQL и LaOQL. Эксперименты проводили на различных БД с различным числом элементов на разных видах запросов. К их числу относятся: выборка объектов; выборка значений атрибутов объектов; выборка с простым условием; выборка со сложным условием; запросы на соединение; сортировка и др. Результаты этих экспериментов показали, что язык LaOQL в среднем на 15 % уступает языку HQL по данным показателям. Вопрос повышения производительности языка LaOQL является темой будущих исследований.

1. Объектная модель данных

Существует несколько формальных подходов к построению объектно-ориентированных моделей данных [6—8], в которых по-разному интерпретируются такие аспекты, как правила формирования схемы БД, наименование и идентификация объектов, связь объектов друг с другом, способы создания объектов и т. д. В контексте представленного подхода будем использовать подход, описанный в стандарте ODMG 3.0 [8], а конкретно следующие элементы описания модели: объект; литерал (*literal*); тип объекта; состояние (атрибуты и отношения); привязка

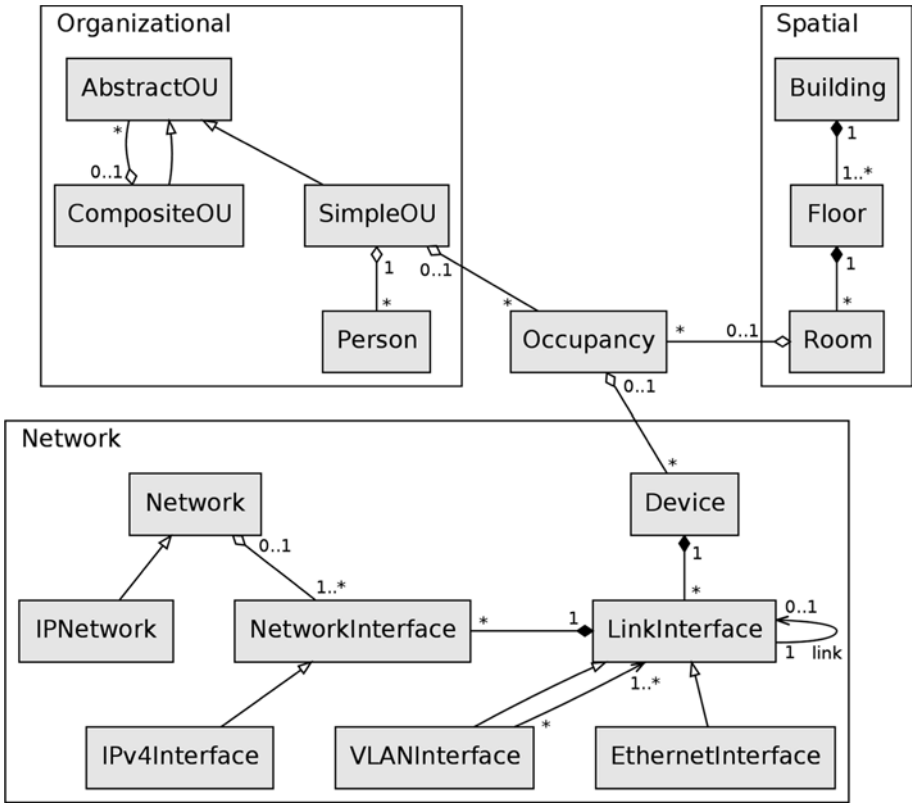

Рис. 1. Модель SON

(*binding*) к языку программирования Java (который используется для реализации языка LaOQL). Рассматриваются следующие виды отношений: наследование; агрегация; композиция. Такие элементы, как идентификация, именование, создание, поведение, привязки к языкам C++ и Smalltalk не используются.

Дальнейшее изложение и примеры запросов будут приведены для ООБД на основе объектно-ориентированной модели SON [9]. Эта модель представляет собой композицию трех моделей структур локального поставщика сетевых услуг (лПСУ): пространственной (S — *Spatial*), организационной (O — *Organizational*) и аппаратно-сетевой (N — *Network*). Модель представлена на рис. 1 в нотации UML 2.0. В табл. 1 определены классы модели SON.

В конкретный момент времени экземпляры классов являются вершинами объектного графа, представляющего архитектуру лПСУ. Дуги графа считаются ненаправленными, строятся согласно связям между классами и интерпретируются по определенному в модели смыслу.

Примеры запросов приведены в табл. 2.

2. Карта предметной области

Идея лаконизации текста запроса заключается в предварительном создании структуры путей, на основе графа классов, полученного из объектной модели с помощью определенных правил (рис. 2). Данную структуру путей назовем картой предметной области (КПО) и будем использовать ее вместо явного задания выражений путей в запросах на поиск в объектном графе.

Правила преобразования диаграммы классов в граф классов. Правила преобразования UML-диаграммы классов в граф классов применяют для различных типов связей: ассоциация (рассматривают следующие типы — простая ассоциация, композиция, агрегация, класс ассоциации и атрибуты — направление и кратность), наследование, реализация. Рассмотрим каждое из них подробно.

- Ассоциация (рис. 3): если класс A имеет отношение с классом B посредством ассоциации через некоторое непустое множество значений атрибутов $P = \{p_1, ..., p_n\}$, то в качестве путей используются переходы по значению каждого атрибута из P.

Общая схема путей будет выглядеть следующим образом: $[[p_1], ..., [p_n]]$, а записанный в КПО путь — $\{A\ \{B\ [[p_1], \ldots, [p_n]]\}\}$. Также приведем пример из модели SON: EthernetInterface → EthernetInterface (запись в КПО: {EthernetInterface {EthernetInterface [[link]]}}).

Рис. 2. Схема лаконизации

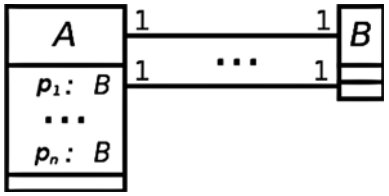

Рис. 3. Правило ассоциации

Таблица 1

Классы модели SON

Имя класса	Объекты, представляемые классом
Building	Здания
Floor	Этажи в здании
Room	Комнаты на этаже
AbstractOU	Обобщенная организационная единица
CompositeOU (COU)	Составные организационные единицы
SimpleOU (SOU)	Простейшие организационные единицы
Person	Сотрудники простейшей организационной единицы
Occupancy (Occup)	Части комнаты, занимаемые простейшими организационными единицами. Определяет пару (Room, SimpleOU), однозначно задающую расположение устройств в пространственной и организационной структурах
Network (UNet)	Сеть ЭВМ
IPNetwork (IPNet)	Сети, использующие технологии протокола IP
NetworkInterface (UNI)	Обобщенный сетевой интерфейс
IPv4Interface (IPv4I)	Интерфейсы IP v. 4
Device	Сетевые устройства
LinkInterface (LI)	Интерфейсы устройства канального уровня
VLANInterface (VLAN)	Интерфейсы виртуальных сетей
EthernetInterface (EI)	Интерфейсы технологии Ethernet

Таблица 2

Примеры запросов

Запрос	Запрос на языке HQL
Найти сетевое устройство в здании главного корпуса	select d from Device as d left join d.occupancies as os where os.room.floor.building.name="Main Building"
Найти маршрутизатор для устройства	select d from Device as d left join d.linkInterfaces as lis left join lis.networkInterfaces as nis left join nis.network.networkInterfaces as nis2 where nis2.linkInterface.device.forwarding=true and d.id=25
Найти все IP-адреса, принадлежащие кафедре ИМО	select ipv4.inetAddress, ei.MACAddress from IPv4Interface ipv4 join ipv4.linkInterface ei where ipv4.linkInterface.device.occupancy.OU.name="Кафедра ИМО"

• Кратность ассоциации (рис. 4): если класс элементов коллекции принадлежит модели данных, то в качестве пути до этого класса используется название коллекции.

Общая схема путей будет выглядеть следующим образом: [[p]], а записанный в КПО путь — {A {B [[p]]}}. Также приведем пример из модели SON: Floor → Room (запись в КПО: {Floor {Room [[rooms]]}}).

• Набор ассоциаций (рис. 5): если класс X_1 имеет связь с классом X_n через набор классов X_2, ..., X_{n-1} посредством значений атрибутов pr_1, ..., pr_{n-1} соответственно, то в качестве пути используются переходы по значениям атрибутов pr_1,..., pr_{n-1}.

Общая схема путей будет выглядеть следующим образом: [[pr_1, ..., pr_{n-1}]], а записанный в КПО путь — {A {B [[pr_1, ..., pr_{n-1}]]}}. Также приведем пример пути из модели SON: SimpleOU → Building (запись в КПО: {SimpleOU {Building [[occupancies room floor building]]}}).

• Наследование и ассоциация с родителем (рис. 6): если класс C наследуется от класса A и класс

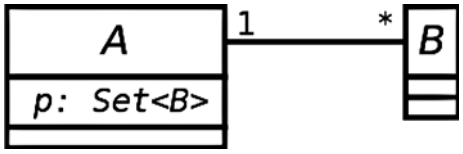

Рис. 4. Правило кратности ассоциации

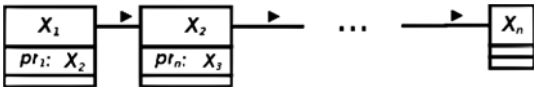

Рис. 5. Правило набора ассоциаций

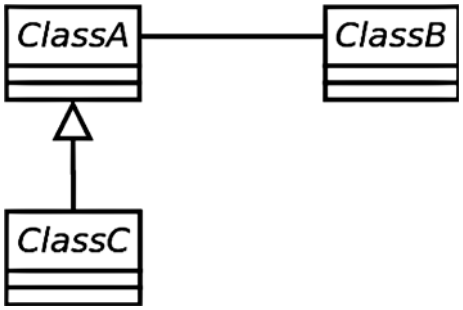

Рис. 6. Правило наследования и ассоциации с родителем

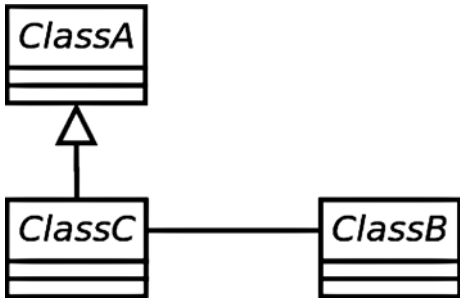

Рис. 7. Правило наследования и ассоциации с потомком

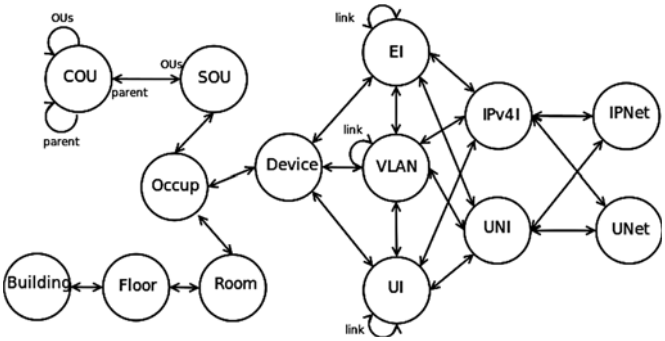

Рис. 8. Граф классов модели SON

A имеет связь с классом B, то при отсутствии прямого пути между классами C и B для пути $C \to B$ ($B \to C$) используется путь $A \to B$ ($B \to A$).

Пример из модели SON: EthernetInterface → Device и Device → EthernetInterface.

Пример КПО модели SON: {EthernetInterface {Device [["device"]]} Device {EthernetInterface [["link Interfaces"]]}}.

- Наследование и ассоциация с потомком (рис. 7): если класс C наследуется от класса A и класс C имеет ассоциацию с классом B, то при отсутствии прямого пути между классами A и B, если объект класса A является объектом класса C, то используется путь от C до B.

Используя данные правила, определим граф классов:

◇ Пусть $C = \{c_1, ..., c_n\}$ — множество классов.

◇ Пусть $assoc = \{<c_n, c_m>, ...\}$ — множество отношений ассоциации, где каждый элемент $<c_n, c_m>$ — это пара классов c_n и c_m, связанных отношением ассоциации. Следует отметить, что если ассоциация двунаправленная, то множество $assoc$ должно содержать два различных элемента: $<c_n, c_m>$ и $<c_m, c_n>$.

◇ Пусть $gen = \{<c_n, c_{n0},..., c_{nm}>, ...\}$ — множество отношений обобщения, где c_n — это родитель, $c_{n0}, ..., c_{nm}$ — это потомки.

Тогда граф классов — это $G_c = <V_c, E_c>$, где V_c — множество вершин, совпадающее с множеством классов (C), а $E_c = \{e_1,..., e_n\}$ множество ребер, упорядоченных пар классов, таких, что либо $e_i \in assoc$, либо $e_i = <c_i, c_j>$, таких, что $\exists <c_k, ..., c_j, ...> \in gen$ (c_j является потомком для некоторого c_k) и $\exists <c_i, c_k> \in assoc$.

Граф объектов — это граф, в котором множеством вершин является множество экземпляров классов (объектов), а множеством дуг — связи, полученные на основе ассоциации между классами диаграммы классов UML. Определим граф объектов следующим образом.

Пусть $objects = \{o_1, ..., o_n\}$ — множество объектов.

Пусть имеется некоторая функция F, отображающая объект в класс: $class(o_i) \in c$, где $o_i \in objects$ и $c \in C$ (c — класс из множества классов).

Тогда граф объектов $G_o = <V_o, E_o>$, где V_o — множество вершин, совпадающее с множеством объектов, а E_o — множество ребер, таких, что для некоторой пары объектов $<o_i, o_j>$ существует отображение $<class(o_i), class(o_j)> \to <c_k, c_n> \in E_c$.

Утверждение о путях: набор путей, построенный на основе графа классов по определенным выше правилам, содержит пути ко всем объектам в графе объектов.

Доказательство. Предположим, что в графе объектов имеется такой путь $<e_k, ..., e_m>$, где $e_i \in E_o$, что применяя функцию отображения F для некоторого e_i, получаем, что результат отображения не содержится в множестве E_c: $<class(o_i), class(o_j)> \to <c_k, c_n> \in E_c$. Очевидно, что это противоречит определению графа объектов, а следовательно, наше предположение неверно.

Пример графа для модели SON, полученного на основе представленных выше правил, изображен на рис. 8.

3. Выражение пути (path-expression)

Выражение пути (*path-expression*) представляет собой одну из базовых концепций объектных языков запросов. Одной из первых работ на данном

направлении исследований является работа [10]. Этот термин можно также встретить в работах [3, 11, 12].

Разные источники определяют это понятие поразному, согласуя его с тем преобразованием объектной модели в граф, которое было выполнено для решения поставленной задачи. Однако можно выделить общую часть, присущую всем определениям, и сформулировать ее следующим образом: выражение пути — это путь в графе объектов.

С формальной точки зрения выражение пути представляет собой последовательность из вершин и дуг графа объектов и может быть представлено следующей конструкцией [13]:

$$[v_0]l_0[v_1]l_1 \ldots [v_{n-1}]l_n[v_n],$$

где v_i — это вершина; l_i — метка дуги из вершины v_{i-1} в вершину v_i.

Одним из методов сокращения текста запроса является сокращение выражения пути. Однако в этом случае возникает вопрос о выборе пути между двумя объектами, когда имеется несколько различных путей. Его разрешению посвящено много исследований [10, 12—15].

Требования к выражению пути в рамках рассматриваемого подхода формулируются следующим образом.

- Задание пути по классам, связанным ассоциацией:

clazz1.clazz2,

clazz1 и *clazz2* соединены прямой связью ассоциации.

- Возможность опускать промежуточные звенья пути:

clazz1.clazz3,

clazz1 и *clazz3* соединены прямой связью ассоциации через промежуточный класс *class2*. Полный путь:

clazz1.clazz2.clazz3.

- Возможность определения рекурсивных путей:

clazz.*

Переход будет осуществляться до тех пор, пока в случае атомарного значения оно не будет равно nil, а в случае списка — пустому списку.

- Возможность указания атрибута в конце пути:

clazz.attr.

В качестве результата будет значение этого атрибута.

- Возможность указания в качестве промежуточного звена атрибута, который имеет класс из модели

clazz.attr.clazz.

Для того чтобы иметь возможность искать пути в КПО, тип объекта *attr* должен представлять собой класс из модели.

- Возможность указания объекта в пути (в нашем случае это реализовано через запросы с параметрами).
- Возможность задания пути к наследникам при отсутствии прямой связи ассоциации между этими классами.

4. Язык запросов LaOQL

В данном разделе описан язык запросов LaOQL, а также решения, с помощью которых можно добиться лаконичности. Основная идея заключается в том, чтобы сократить запись выражения пути. Однако, так как эта достаточно сложная задача, то мы не пытаемся ее решить в общем, а ограничиваемся объектно-ориентированной моделью, описанной в разд. 1.

Алгоритм построения КПО можно описать следующим образом.

```
1: CLASSES — список классов
2: DOM — отображение, представляющее собой КПО
3: get-paths — функция получения списка путей между двумя классами
4: for all source ∈ CLASSES do
5:     for all target ∈ CLASSES do
6:         paths ← (get-paths source target)
7:         DOM ← (assoc-in DOM [source target] paths)
8:     end for
9: end for
```

Таблица 3

Свойства запросов языка LaOQL

Запрос	Выражение
Выборка объектов определенного класса	T
Обращение к атрибуту	$T.p_k$, где $1<=k<=n$
Подмножество атрибутов класса (например, атрибуты p_k и p_t, где $1<=k<=n$ и $1<=t<=n$)	$T[p_k\ p_t]$
Условие на значение атрибутов (знак #). В определении условий поддерживаются операции дизъюнкции (||), конъюнкции (&&) и операции отношений (=,< =,> =,<,>,~, а также комбинация из символа ! и любого из перечисленных знаков, означающая отрицание полученного значения)	$T\#(p_k=1)$ $T\#(p_k=1 \mid\mid p_k=2)$ $T[p_k]\ \#(p_k=1 \mid\mid p_k=2)$
Объединение результатов запроса	$Q_1, \ldots, Q_n$
Выборка связанных объектов	$T_1(T_2),\ T_1(T_2,T_3)$

Таблица 4

Примеры запросов на языке LaOQL

Запрос	Запрос на языке LaOQL
Найти сетевое устройство в здании главного корпуса	d#(b=Main Building)
Найти маршрутизатор для устройства	d (ni#(d.forwarding=true && d.id=25))
Найти все IP-адреса, принадлежащие кафедре ИМО	(ipv4[inetAddress], ei[MACAddress])#(ou.name="Кафедра ИМО")

Язык LaOQL представляет собой способ задания выборки объектов и оперирует для этого именами классов этих объектов и их атрибутами. Свойства запросов этого языка представлены в табл. 3, где рассматривается класс модели с именем T и набором атрибутов $p_1 \ldots p_n$.

Примеры запросов на языке LaOQL, эквивалентные запросам табл. 2, приведены в табл. 4.

Как видно из представленных примеров, авторское решение позволяет существенно сократить текст запроса.

5. Реализация языка

Прототип языка LaOQL был реализован с помощью языков Java и Clojure [15]. Для синтаксического и лексического анализа применялись инструменты JFlex и byaccj соответственно. Язык Clojure использовался для работы с КПО и обращении к источникам данных. Высокоуровневая архитектура реализации языка представлена на рис. 9. Количественные характеристики кода представлены в табл. 5.

На настоящее время язык можно использовать с ООБД и системами, которые реализуют Java Persistence API [16], такие как Hibernate, OpenJPA, TopLink, ObjectDB, db4o и др.

На рис. 10 (см. вторую сторону обложки) представлен интерфейс работы с языком запросов в рамках проекта Nest [9]. Пользователь вводит текст запроса в текстовое поле, а в качестве результата получает таблицу с результатом его выражения, а также используемый в ходе выполнения запроса путь. Ячейки таблицы представляют собой интерактивные объекты, с которыми можно выполнять различные действия (переход на граф визуализации, переход на редактирование объекта и др.).

Таблица 5

Количественные характеристики кода

Характеристика	Используемые языки			
	JFlex	byaccj	Java	Clojure
Строки кода	379	544	1338	3095
Число модулей	1	1	1	8

Тестирование. Для тестирования языка запросов LaOQL использовали систему JUnit. Для основных конструкций языка создавались и автоматически генерировались графы объектов модели SON. Например, если тестовым испытаниям подвергалась выборка с условиями, то у объектов заполнялись значения атрибутов для возможности проверки правильности полученного результата. Каждый тест автоматически сравнивал результат запроса с корректным значением, которое задавали вручную. Всего было создано порядка 30 графов объектов и 500 тестов.

Эксперименты. Для определения производительности выполнения запроса проводили ряд экспериментов. С этой целью была создана система генерации графа объектов модели SON, которая основывалась на весовых характеристиках для каждого класса объектов. Например, на каждое здание должно приходиться пять этажей и 1000 устройств. Заполнялись также атрибуты объектов из заранее определенного набора значений. Функция генерации графа объектов модели SON получала на вход общее число элементов в графе и возвращала полученный граф.

В ходе эксперимента измеряли время между передачей на вход лексического анализатора текста запроса и получением результата. Эксперимент проводили заданное число раз, из полученных результатов удаляли самый большой и самый маленький результаты. На основе остальных результатов вычисляли среднее значение, которое принимали в качестве результата эксперимента.

Рис 9. Архитектура реализации языка LaOQL

Таблица 6

Результаты экспериментов: время выполнения запроса, с

БД	Язык	Число объектов						
		1000	5000	10 000	15 000	20 000	50 000	100 000
Derby	HQL	1,2	1,5	2,1	5,3	5,6	12,4	34,1
	LaOQL	1,3	1,9	3,0	8,6	10,1	15,6	50,1
H2	HQL	1,5	1,5	2,3	3,1	7,1	15,1	40,5
	LaOQL	1,5	1,6	2,5	6,4	8,5	20,4	60,1
HSQLDB	HQL	0,8	1,0	1,3	2,5	3,4	10,1	17,3
	LaOQL	0,9	1,0	1,7	4,9	6,1	14,5	31,2

Эксперименты проводили на различных базах данных (Derby, H2, HSQLDB) с различным числом элементов в графе объектов (1000, 5000, 10 000, 15 000, 20 000, 50 000, 100 000).

В табл. 6 представлены результаты экспериментов для списка из различных 15 запросов. Как видно из данных таблицы, производительность LaOQL меньше производительности языка HQL.

Заключение

Разработан язык запросов, который позволяет существенным образом сократить текст запроса по сравнению с существующими аналогами. Для решения этой задачи предложены следующие механизмы: возможность не указывать промежуточные пути, значение атрибута по умолчанию, сокращение имен классов, сокращение сложных условий.

Для решения вопроса о выборе пути предложена карта предметной области, которая строится общим алгоритмом поиска в ширину, а затем редактируется вручную поставщиком системы исходя из содержательных соображений о предметной области.

Дальнейший интерес представляют исследования в направлении улучшения производительности. Особого внимания заслуживают изучение возможностей параллельного выполнения алгоритма реализации запроса и агрегации результатов обработки по уже полученным ранее результатам.

Параллельное выполнение может быть применено в следующих случаях: проверка объектов на удовлетворение набору условий, переход по связям, выполнение функций. Есть все основания предполагать, что данное внедрение существенным образом увеличит производительность существующей реализации языка LaOQL.

Список литературы

1. **Date C.** An Introduction to Database Systems. 8 edition. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

2. **Эльдарханов А.** Обзор моделей данных объектно-ориентированных СУБД // Труды Института системного программирования РАН. 2011. Т. 21. С. 205—226.

3. **Bertino E., Negri M., Pelagatti G., Sbattella L.** Object-oriented query languages: The notion and the issues // IEEE Transactions on Knowledge and Data Engineering. 1992. Vol. 4, No. 3. P. 223—237.

4. **Kulkarni K. G.** Object-Oriented Extensions in SQL3: A Status Report Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24—27, 1994 / Ed. By R. T. Snodgrass, M. Winslett. ACM Press, 1994. 478 p.

5. **Bauer C.** Hibernate in Action (In Action series). Greenwich, CT, USA: Manning Publications Co., 2004.

6. **Cattell R., Barry D. K.** The Object Data Standard: ODMG 3.0. San Francisco, California: Morgan Kaufmann, 2000.

7. **Cacace F., Ceri S., Crespi-Reghizzi S.** Integrating Object-oriented Data Modelling with a Rule-based Programming Paradigm // ACM SIGMOD Record. 1990. Vol. 19. P. 225—236.

8. **Rumbaugh J., Jacobson I., Booch G.** Unified Modeling Language Reference Manual, the (2nd Edition). Pearson Higher Education. 2004.

9. **Богоявленский Ю. А.** Прототип экспериментальной платформы Nest для исследования моделей и методов управления ИКТ-инфраструктурами локальных поставщиков услуг Интернет // Программная инженерия. 2013. № 2. С. 11—20.

10. **Kifer M., Kim W., Sagiv Y.** Querying Object-Oriented Databases // ACM SIGMOD International conference on management of data. ACM New York, 1992. P. 393—402.

11. **Lee D., Chien W.** Using Path Information for Query Processing in Object-Oriented Database Systems // Proceedings of Conference on Information and Knowledge Management. 1994. P. 64—71.

12. **Ioannidis Y. E., Lashkari Y.** Incomplete Path Expressions and Their Disambiguation // SIGMOD Rec. 1994. May. Vol. 23, Iss. 2. P. 138—149.

13. **Lieberherr K.** Navigating through Object Graphs Using Local Meta-Information: Tech. rep. W: 2001.

14. **Bussche J. V. D., Vossen G.** An Extension of Path Expressions to Simplify Navigation in Object-Oriented Queries // In Proc. of Intl. Conf. on Deductive and Object-Oriented Databases (DOOD). Springer, 1993. P. 267—282.

15. **Mehta A., Geller J., Perl Y., Neuhold E.** The OODB Path-Method Generator (PMG) Using Access Weights and Precomputed Access Relevance // The VLDB journal. 1998. Vol. 7, Iss. 1. P. 25—47.

16. **Burke B., Monson-Haefel R.** Enterprise JavaBeans 3.0 — Developing Enterprise Java Components: Covers Java Persistence. 5. ed. O'Reilly, 2006.

Laconic Object Query Language Using Features of Object Model

V. M. Dimitrov, dimitrov@cs.petrsu.ru, **A. V. Voronin**, voronin@petrsu.ru,
Iu. A. Bogoiavlenskii, ybgv@cs.petrsu.ru, Petrozavodsk State University,
Petrozavodsk, 185910, Russian Federation

Corresponding author:

Dimitrov Vycheslav M., Senior Lecturer, Petrozavodsk State University, Petrozavodsk, 185910, Russian Federation
E-mail: dimitrov@cs.petrsu.ru

Received on December 22, 2016
Accepted on December 27, 2016

The article presents a concise query language — Laconic Object Query Language (LaOQL) that allows one to reduce text length of a query to object-oriented database (OOBD), and thus the time for its preparation. For example, if an object-oriented database OOBD contains information about the network, this intensive sending of queries to the data base is required for searching for sources of traffic anomalies, the studying of configuration, and for solving of other similar network management tasks. In such circumstances, the brevity of the query text can significantly improve productivity of an OOBD user (in this example, an administrator or a network researcher) and reduce the possibility of errors in the query text, as well. Our ideas for reaching concise are: removing writing links between object entities; using laconic names for object entities; using the default property values; using laconic notation for complex conditions.

Keywords: *object-oriented database system, query language, network*

For citation:

Dimitrov V. M., **Voronin A. V., Bogoiavlenskii Iu. A.** Laconic Object Query Language Using Features of Object Model, *Programmnaya Ingeneria*, 2017, vol. 8, no. 3, pp. 112—119.

DOI: 10.17587/prin.8.112-119

References

1. **Date C.** *An Introduction to Database Systems.* 8 edition, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

2. **Eldarhanov A.** Obzor modelej dannyh ob#ektno-orientirovannyh SUBD (Review of OODM models), *Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS)*, 2011, vol. 21, pp. 205—226 (in Russian).

3. **Bertino E., Negri M., Pelagatti G., Sbattella L.** Object-oriented query languages: The notion and the issues, *IEEE Transactions on Knowledge and Data Engineering,* 1992, vol. 4, no. 3, pp. 223—237.

4. **Kulkarni K. G.** *Object-Oriented Extensions in SQL3: A Status Report Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, May 24—27, 1994 / Ed. By R. T. Snodgrass, M. Winslett, ACM Press, 1994, 478 p.

5. **Bauer C.** *Hibernate in Action (In Action series),* Greenwich, CT, USA: Manning Publications Co., 2004.

6. **Cattell R., Barry D. K.** *The Object Data Standard: ODMG* 3.0, San Francisco, California: Morgan Kaufmann, 2000.

7. **Cacace F., Ceri S., Crespi-Reghizzi S.** Integrating Object-oriented Data Modelling with a Rule-based Programming Paradigm, *ACM SIGMOD Record*, 1990, vol. 19, pp. 225—236.

8. **Rumbaugh J., Jacobson I., Booch G.** *Unified Modeling Language Reference Manual*, the (2nd Edition), Pearson Higher Education, 2004.

9. **Bogoiavlenskii Iu. A.** Prototip jeksperimental'noj platformy Nest dlja issledovanija modelej i metodov upravlenija IKT-infrastrukturami lokal'nyh postavshhikov uslug Internet (Prototype of the Testbed Nest for Research of Network Management Methods and Models at the Enterprise Network Level), *Programmnaya Ingeneria*, 2013, no. 2, pp. 11—20 (in Russian).

10. **Kifer M., Kim W., Sagiv Y.** Querying Object-Oriented Databases, *ACM SIGMOD International conference on management of data*, ACM New York, 1992, pp. 393—402.

11. **Lee D., Chien W.** Using Path Information for Query Processing in Object-Oriented Database Systems, *Proceedings of Conference on Information and Knowledge Management*, 1994, pp. 64—71.

12. **Ioannidis Y. E., Lashkari Y.** Incomplete Path Expressions and Their Disambiguation, *SIGMOD Rec.,* 1994, vol. 23, iss. 2, pp. 138—149.

13. **Lieberherr K.** Navigating through Object Graphs Using Local Meta-Information: Tech. rep. W: 2001.

14. **Bussche J. V. D., Vossen G.** An Extension of Path Expressions to Simplify Navigation in Object-Oriented Queries, *In Proc. of Intl. Conf. on Deductive and Object-Oriented Databases (DOOD),* Springer, 1993, pp. 267—282.

15. **Mehta A., Geller J., Perl Y., Neuhold E.** The OODB Path-Method Generator (PMG) Using Access Weights and Precomputed Access Relevance, *The VLDB journal,* 1998, vol. 7, iss. 1, pp. 25—47.

16. **Burke B., Monson-Haefel R.** *Enterprise JavaBeans 3.0 — Developing Enterprise Java Components: Covers Java Persistence.* 5. ed., O'Reilly, 2006.

УДК 004.72 DOI: 10.17587/prin.8.120-128

Д. К. Левоневский, науч. сотр., e-mail: DLewonewski.8781@gmail.com,
И. В. Ватаманюк, аспирант, e-mail: vatamaniuk@iias.spb.su, **А. И. Савельев,** науч. сотр., e-mail: antoni-fox@yandex.ru, Санкт-Петербургский институт информатики и автоматизации РАН

Многомодальная информационно-навигационная облачная система МИНОС для корпоративного киберфизического интеллектуального пространства

Рассмотрена архитектура корпоративной многомодальной информационно-навигационной облачной системы (МИНОС), описаны ее компоненты и базовые положения работы подсистемы корпоративного телевидения. Представлены инфраструктурные компоненты, необходимые для эффективной эксплуатации системы, включая четыре стационарных монитора, четыре неттопа, сетевое оборудование, веб-сервер, сервер базы данных. Клиентское и серверное программное обеспечение для управления системой корпоративного телевидения обеспечивает администрирование и поддержку актуального контента через веб-интерфейс. Рассмотрены сценарии использования системы для различных категорий пользователей, а также принципы управления контентом. Такой контент отображается на экране с помощью специальных сигналов и событий, которые генерируются по временным меткам или по результатам взаимодействия с пользователем.

Ключевые слова: *киберфизические системы, многомодальные интерфейсы, корпоративные информационные системы, информационно-навигационные системы, облачные системы*

Введение

Повсеместное распространение киберфизических систем в настоящее время обусловлено их широкими возможностями для организации интерактивных информационных сред, в том числе корпоративных систем, предоставляющих информационно-навигационные услуги для посетителей. Многомодальные средства коммуникации, предоставляемые киберфизическими системами, значительно превосходят возможности традиционных справочных систем. Они позволяют реализовать взаимодействие пользователей с системой, используя жесты, речь, движения и т. д., что упрощает работу с ней неподготовленным пользователям [1—3]. В статье рассмотрены результаты разработки и опыт применения подобного подхода для организации корпоративной информационно-навигационной облачной системы.

Распространенным способом построения корпоративных информационно-навигационных систем является использование технологии *Digital Signage* (DS). В отсутствие общепринятого перевода чаще всего данный термин переводят как "цифровые вывески". Как правило, данная технология используется для распространения рекламного контента. Однако спектр ее возможностей не ограничивается маркетингом, предоставляя широкий потенциал в сфере образования, корпоративных коммуникаций, многопользовательских систем ведения переговоров и т. п. [4, 5]. Далее рассмотрим несколько примеров использования технологии DS в информационных системах различного назначения.

1. Анализ систем интеллектуального информационного окружения

Как правило, информационные системы на основе технологии DS состоят из нескольких экранов, объединенных в сеть [6]. Особый интерес представляют технологии управления подобными системами, которые могут использовать как стандартные веб-технологии [7, 8], так и интерактивное взаимодействие с сервисами Интернета вещей [9, 10], пользовательскими устройствами [11], а также распознавание лица пользователя [12], распознавание жестов, артикуляции [13, 14] и т. п. Рассмотрим подробнее некоторые существующие разработки в данной области.

В работе [15] рассмотрена концепция применения технологии DS в области управления энергопотреблением зданий (BEMS). Применение технологий игрофикации в совокупности с системой DS позволяет заинтересовать пользователей в отслеживании потребления ресурсов и экономии электроэнергии. Технология DS используется для визуального пред-

ставления данных об энергопотреблении организации в виде структурированной информации, наглядной, легкой для понимания и привлекающей повышенное внимание.

В работе [12] предложена технология взаимодействия систем DS и мобильных устройств пользователей. Подобная интерактивность позволяет пользователям потреблять распространяемый системой аудиовизуальный контент. Она делает более доступной полезную информацию и предоставляет возможности для сохранения данных, удобного перехода по внешним ссылкам в сеть Интернет, для интерактивной навигации в здании. Подобные системы показывают хорошую масштабируемость [16], позволяя создавать так называемые экосистемы программного обеспечения — сложные программные платформы, которые разрабатывают и поддерживают децентрализованно.

Использование технологий DS в различных системах позволяет создавать киберфизическую интерактивную дополненную реальность [7, 17, 18]. Архитектура программно-конфигурируемой интеллектуальной системы DS на основе технологии Интернета вещей [19] предложена в работе [11]. Разработанная авторами система основана на использовании коммуникации посредством невидимых датчиков изображения (ISC). В них используются механизмы для встраивания данных в видеоизображение с помощью "водяных знаков", незаметных для человеческого глаза, но детектируемых посредством камеры. Система DS взаимодействует с пользователями посредством камеры смартфона и может подключать пользователей к облачной сети, используя динамические протоколы. Работа в интерактивном окружении данной интеллектуальной системы осуществляется одновременно с демонстрацией контента, что повышает удобство пользования системой при увеличении числа пользователей.

Еще один шаг вперед сделан в направлении интеграции пользовательских приложений и систем DS в работе [20], где предложена концепция динамической адаптации аудиовизуального контента к окружению посредством использования публичных приложений. Такие приложения позволяют участвовать в формировании расписания дисплеев не только администраторам системы, но и пользователям. Этот подход позволяет повысить не только актуальность отображаемой информации, но и персонализацию контента.

2. Архитектура информационно-навигационной системы МИНОС

Корпоративная информационно-навигационная система МИНОС состоит из четырех частей, описанных далее (рис. 1).

1. Система идентификации, выполняющая процедуру определения посетителей организации на входе по их ID-картам (для сотрудников) или с помощью алгоритмов распознавания лиц (для других гостей). Эта система также хранит профили посетителей (личные и контактные данные, фотографии), она предоставляет посетителям доступ к информационно-навигационным и справочным сервисам через меню.

2. Система информационной поддержки, взаимодействующая с пользователями с помощью стационарных камер и экранов, расположенных в разных местах организации. В ее функции входит трансляция

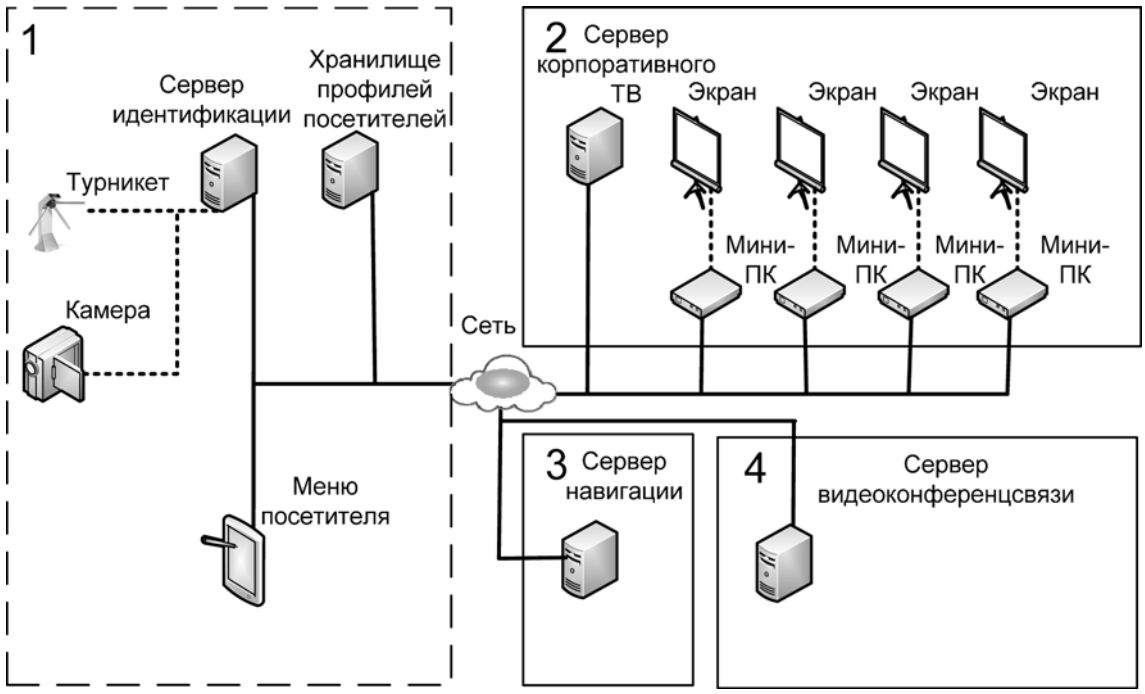

Рис. 1. Архитектура МИНОС

на стационарные экраны различной информации для сотрудников и посетителей (сведения об институте и его деятельности, демонстрация разработок, объявления, приветствия, поздравления) по их запросу и/или в соответствии с расписанием.

3. Система навигации, в функции которой входит предоставление сотрудникам и посетителям справочной информации (расположение посетителя и подразделений организации, карты и модели этажей) по QR-кодам, расположенным в здании. Сервер навигации предоставляет пользователям веб-приложение, с которым они могут взаимодействовать с помощью портативного компьютера или мобильных устройств.

4. Система видеоконференцсвязи, позволяющая посетителям связываться со структурными подразделениями организации [21, 22].

На данный момент в Санкт-Петербургском институте информатики и автоматизации Российской академии наук (СПИИРАН) выполнена разработка системы корпоративного телевидения как компонента МИНОС. Рассмотрим архитектурно-технологические характеристики системы более подробно.

3. Система корпоративного телевидения

Система корпоративного телевидения обеспечивает трансляцию на стационарные экраны (клиентские системы) информации в соответствии с расписанием, которое обновляется сервером динамически на основе множества активных событий. Под экраном понимается стационарный программно-аппаратный комплекс, предназначенный для отображения медиаконтента. Он состоит из дисплея и неттопа, подключенного к серверу информационной системы. Для работы монитора на неттоп под управлением ОС Debian или Ubuntu устанавливается контроллер клиентского программного обеспечения, управляющий компонентами, необходимыми для работы МИНОС. К их числу относятся: веб-браузер с поддержкой HTML5, на котором работает клиентское веб-приложение; система идентификации посетителей; система управления экраном. Архитектура клиентской части системы представлена на рис. 2.

К компонентам серверной части относятся: база данных, хранящая информацию о медиаконтенте, событиях, мониторах и расписаниях; сервер приложений, выполняющий обновление расписаний и формирующий ответы на запросы клиентов; веб-сервер, предоставляющий клиентам информацию через сеть. Архитектура сервера представлена на рис. 3.

Упрощенная диаграмма последовательности, описывающая взаимодействие клиентского веб-приложения и сервера, представлена на рис. 4. Структурными компонентами веб-приложения являются контроллер (Controller) и модуль представления (Viewer). Контроллер состоит из модулей передачи

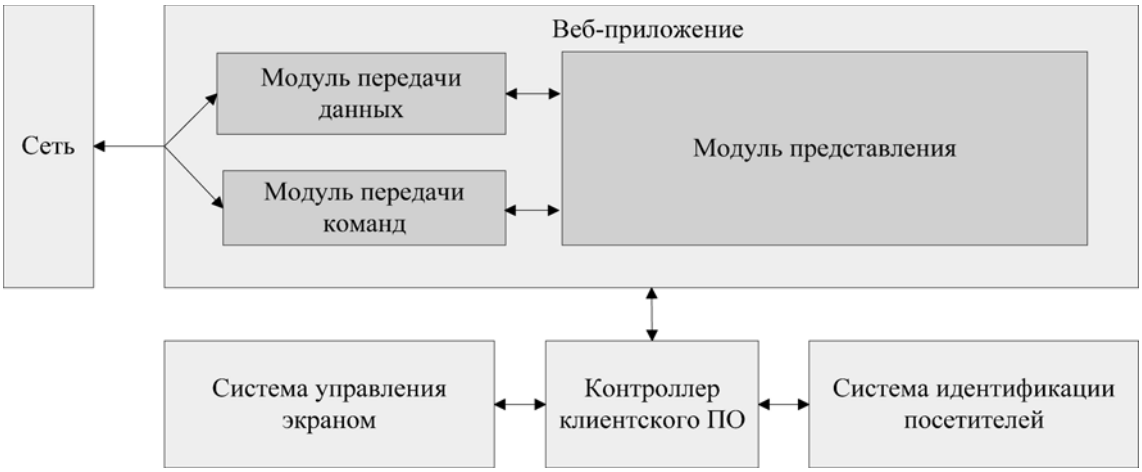

Рис. 2. Архитектура клиентской части системы

Рис. 3. Архитектура сервера

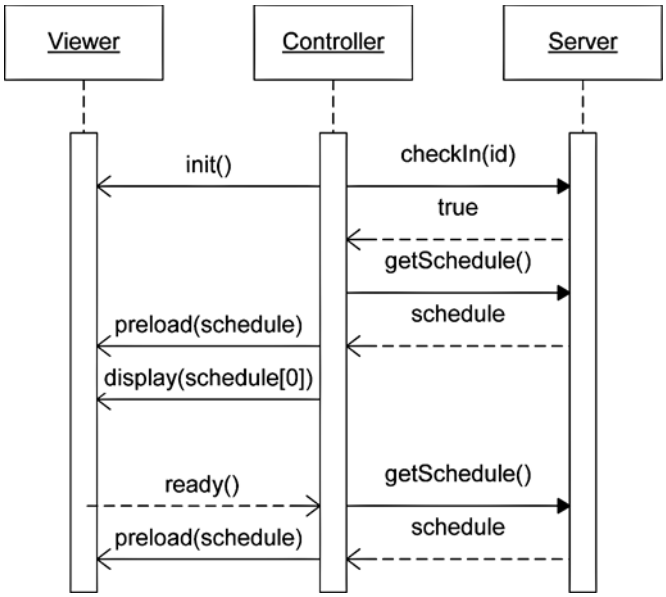

Рис. 4. Диаграмма последовательности для монитора

данных и команд. Он соединяется с сервером (Server), получает информацию о расписании и передает ее модулю просмотра. Модуль просмотра выполняет предзагрузку медиафайлов и их отображение, а также сообщает контроллеру о готовности воспроизводить следующий файл.

В системе корпоративного телевидения каждый экран характеризуется идентификатором, описанием, ключом доступа, режимом работы и имеет свое расписание. Экраны можно объединять в группы и управлять всеми экранами группы так же как и одиночным экраном. Экраны отключены в нерабочее время (отключение выполняется автоматически) и во время обслуживания (отключение выполняется оператором). В другое время они включены. Рассматривается также вариант автоматического включения и выключения экранов в зависимости от присутствия пользователей. Для этого применяют технологии компьютерного зрения.

Управление контентом, отображаемым на экране, выполняется с помощью специальных сигналов и событий.

Сигнал — сообщение, которое отправляется оператором или системой экрану и предписывает экрану транслировать определенный контент. Этот способ используется, например, для отображения приветствий сотрудникам организации (в этом случае источником сигнала служит система идентификации посетителей) или трансляции объявлений (объявления создаются вручную оператором).

Событие — объект, который характеризуется состоянием (активно или неактивно), множеством экранов, связанных с событием, и множеством медиафайлов, ассоциированных с ним. Если событие активно, то ассоциированные медиафайлы автоматически добавляются в расписание экранов, связанных с событием. Таким способом выполняется трансляция информации о деятельности организации, проведении конференций и выставок, праздниках и т. д.

Условия активности события задают интервалы времени, в течение которых выполняется трансляция. Для задания интервалов используют обозначения из табл. 1, которые соответствуют наиболее часто употребляемым параметрам даты и времени

Таблица 1

Обозначения, используемые для задания временных интервалов

Обозначение	Описание	Обозначение	Описание
[D]	День месяца (1...31)	[DY]	День в году (1...366)
[M]	Месяц (1...12)	[DW]	День недели (1-Пн, 2-Вт, ...)
[Y]	Год	[TH]	Час (0...23)
[DM]	Дата в формате ДД.ММ	[TM]	Минута (0...59)
[DMY]	Дата в формате ДД.ММ.ГГГГ	[TS]	Секунда (0...59)

Таблица 2

Операции для задания условий

Операция	Описание	Операция	Описание
=	Равно	>=	Больше или равно
<>	Не равно	AND	Логическое И
<	Меньше	OR	Логическое ИЛИ
>	Больше	NOT	Логическое НЕ
<=	Меньше или равно	IN	Принадлежность множеству

Таблица 3

Примеры условий выполнения событий

Условие	Описание
([DW] IN (1,2,3,4) AND [TH]>=9 AND [TH]<18) OR ([DW]=5 AND [TH]>=9 AND [TH]<17)	Рабочее время (с 9:00 до 18:00 с понедельника по четверг и с 9:00 до 17:00 в пятницу)
[TH]=13 AND [TM] <=48	Время обеда (с 13:00 до 13:48)
[DY]=256	День программиста (256-й день в году)
[DM]=`08.02`	День российской науки (8 февраля)

и позволяют задать все типовые интервалы. Для задания условий используется инфиксная запись, включающая операции из табл. 2. Примеры условий приведены в табл. 3.

Основной функцией сервера является динамическое формирование расписания медиафайлов для экрана. Расписание представляет собой упорядоченный список (очередь). Новые медиафайлы добавляются в конец очереди по мере того, как медиафайлы из его начала воспроизводятся на экранах и исключаются из очереди.

Алгоритм обновления расписания (т. е. добавления в него N очередных медиафайлов) для экрана M состоит из перечисленных далее шагов [23].

Шаг 1. Определение общего множества активных событий путем проверки их условий активации.

Шаг 2. Определение множества активных событий $\{E_M\}$, соответствующих монитору M.

Шаг 3. Определение множеств активных событий $\{E_{G_i}\}$, соответствующих группам мониторов $G_1, G_2, ..., G_n$, в которых состоит монитор M.

Шаг 4. Определение полного множества активных событий для монитора:

$$\{E\} = \{E_M\} \cup \{E_{G_1}\} \cup \{E_{G_2}\} \cup ... \cup \{E_{G_n}\}.$$

Шаг 5. Исключение из расписания медиафайлов, ассоциированных с событиями, не входящими в множество $\{E\}$.

Шаг 6. Определение множества медиафайлов $\{F\}$, ассоциированных с множеством $\{E\}$.

Шаг 7. Упорядочивание множества $\{F\}$ по времени последней активизации (в порядке возрастания).

Шаг 8. Добавление первых N файлов из множества $\{F\}$ в конец расписания.

Шаг 9. Обновление времени последней активизации для файлов из множества $\{F\}$.

Шаг 10. Ожидание следующего запроса на обновление.

Далее рассмотрим более подробно функциональные возможности корпоративной информационно-навигационной системы, использующей технологию DS, на примере разрабатываемой системы СПИИРАН [23, 24].

4. Сценарии функционирования системы МИНОС

Пользователи, взаимодействующие с компонентами МИНОС, подразделяются на три основные категории: сотрудники института, студенты и гости института (представители делегаций, экскурсанты музея института, одиночные гости, разнорабочие и т. д.) Сценарии использования информационно-навигационной системы представлены в виде диаграммы Use Case на рис. 5.

Взаимодействие системы с посетителем осуществляется с помощью меню, где в зависимости от категории посетителя предлагаются те или иные сервисы. Новые постоянные пользователи должны пройти регистрацию в системе, сдав биометрические данные. Гости института могут обратиться к справочным сервисам, включающим предоставление информации об институте и его структуре, поиск аудиторий и связь с сотрудниками института с помощью системы видеоконференцсвязи [25].

Сотрудники СПИИРАН проходят через турникет, установленный на входе в институт, после автоматической идентификации на основе распознавания лица и индивидуального RFID-пропуска. Остальным категориям пользователей предлагается выбрать соответствующий их статусу пункт меню на сенсорном экране и следовать дальнейшим указаниям. Опции, предлагаемые посетителям через меню, включают, помимо предоставления стандартной справочной информации, возможности звонка сотруднику СПИИРАН с помощью системы видеоконференцсвязи, обращения к навигационному сервису для поиска структурного подразделения или аудитории, вызова робота-гида.

Сценарии многомодальных сервисов регистрации сотрудников института и студентов представлены на рис. 6, *а* и *б* соответственно. Меню, предлагаемое студентам показано на рис. 7, *а*; меню, предлагаемое представителям делегаций, одиночным гостям и другим посетителям, представлено на рис. 7, *б*.

В дальнейшем предполагается разработка сценариев взаимодействия посетителей с экранами корпоративного телевидения, что позволит сделать трансляции персонализированными.

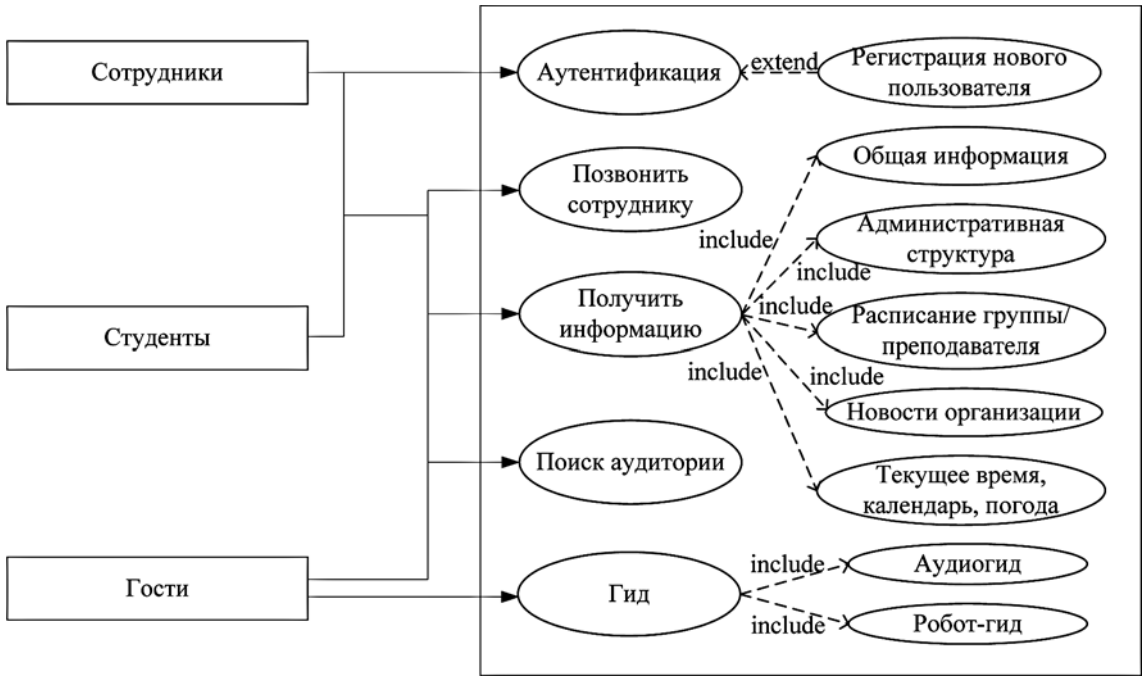

Рис. 5. Сценарии использования информационно-навигационной системы

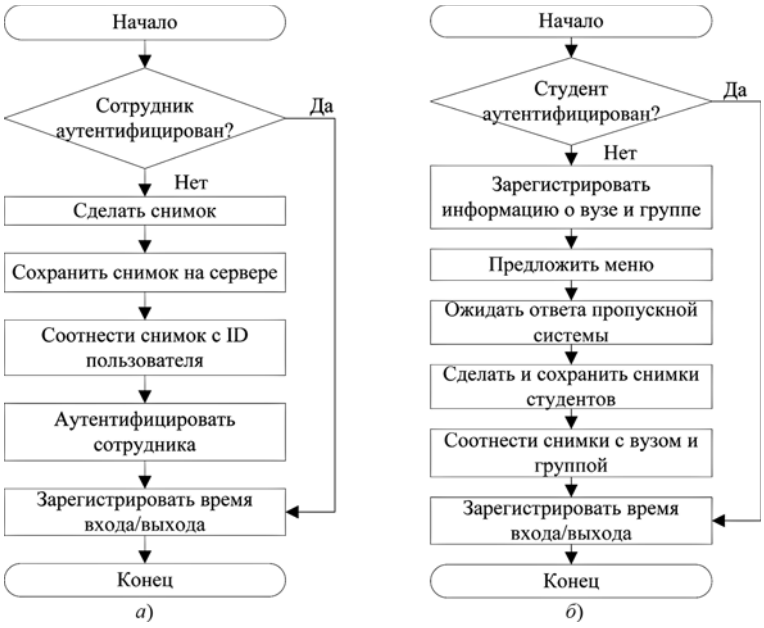

Рис. 6. Сценарии регистрации пользователей:
а — сотрудников; *б* — студентов

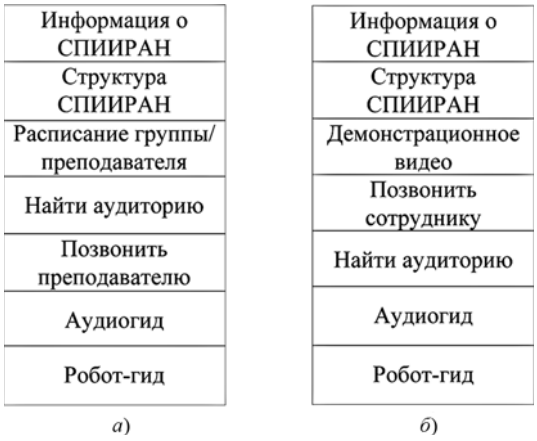

Рис. 7. Меню для посетителей:
а — для студентов; *б* — для остальных посетителей

Заключение

В настоящее время система МИНОС находится на стадии внедрения. Для реализации необходимой инфраструктуры в СПИИРАН использованы четыре стационарных монитора, четыре неттопа, сетевое оборудование, веб-сервер, сервер базы данных. Разработано клиентское и серверное программное обеспечение для управления системой корпоративного телевидения, реализован веб-интерфейс администратора этой системы. Система корпоративного телевидения использует стандартные сетевые технологии, не привязана к конкретным программно-аппаратным платформам. Она соответствует критериям

расширяемости и переносимости и может использоваться в качестве компонента киберфизической среды в различных организациях.

Дальнейшим направлением разработки является обеспечение интерактивности и персонализации системы корпоративного телевидения путем ее интеграции с системой биометрической и RFID-идентификации пользователей. Для этого клиентские станции будут оснащены камерами и микрофонами. Для каждого сотрудника института, распознанного стационарной камерой, система корпоративного телевидения может отображать объявления, имеющие отношение к этому сотруднику. Предполагается реализовать управление выдачей информации с помощью жестов и речи [26, 27].

Другим направлением развития системы является создание навигационного сервиса, который позволяет посетителю определить его местоположение в институте с помощью сканирования табличек с QR-кодами. Веб-приложение позволяет посетителю ориентироваться в институте, используя портативный компьютер, и получать необходимую справочную информацию.

Исследование выполнено при частичной поддержке бюджетной темы № 0073-2015-0005 и РНФ (грант № 15-18-00047).

Список литературы

1. **Ронжин Ан. Л., Ватаманюк И. В., Ронжин Ал. Л., Железны М.** Математические методы оценки размытости изображения и распознавания лиц в системе автоматической регистрации участников совещаний // Автоматика и телемеханика. 2015. № 11. С. 132—144.

2. **Глазков С. В., Ронжин А. Л.** Контекстно-зависимые методы автоматической генерации многомодальных пользовательских веб-интерфейсов // Труды СПИИРАН. 2012. № 2. С. 170—183.

3. **Ронжин А. Л., Карпов А. А., Леонтьева Ан. Б., Костюченко Б. Е.** Разработка многомодального информационного киоска // Труды СПИИРАН. 2007. № 5. С. 227—245.

4. **Han S., Kim N., Choi K., Won K. J.** Design of multi-party meeting system for interactive collaboration // 2nd International Conference on Communication Systems Software and Middleware. IEEE. 2007. P. 1—8.

5. **Kelsen K.** Unleashing the power of digital signage: content strategies for the 5th screen. CRC Press, 2012.

6. **She J., Crowcroft J., Fu H., Ho P.-H.** Smart signage: An interactive signage system with multiple displays // Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing IEEE, 2013. P. 737—742.

7. **Davies N., Clinch S., Alt F.** Pervasive displays: understanding the future of digital signage // Synthesis Lectures on Mobile and Pervasive Computing. 2014. Vol. 8, No. 1. P. 1—128.

8. **Ahn S., Song S., Yang J.** et al. Web-based multi-screen digital signage // IEEE 3rd Global Conference on Consumer Electronics (GCCE). IEEE, 2014. P. 244—245.

9. **Hossain M. A. Lee Y. T., Lee H.** et al. A Symbiotic Digital Signage system based on display to display communication // Seventh International Conference on Ubiquitous and Future Networks. IEEE. 2015. P. 124—128.

10. **Hossain M. A., Islam A., Le N. T.** et al. Performance Analysis of Smart Digital Signage System Based on software-defined IoT and Invisible Image Sensor Communication // International Journal of Distributed Sensor Networks. 2016. Vol. 12, No. 7. P. 1—14.

11. **Kim E., Lee H. J., Lee D. H.** et al. Efficient contents sharing between digital signage system and mobile terminals // Advanced Communication Technology (ICACT), 15th International Conference on. IEEE, 2013. P. 1002—1005.

12. **Farinella G. M., Farioli G., Battiato S.** et al. Face re-identification for digital signage applications // International Workshop on Video Analytics for Audience Measurement in Retail and Digital Signage. Springer International Publishing, 2014. P. 40—52.

13. **Lee Y. T., Lien C. H., Hung A.** et al. Design of a low cost interactive digital signage // The 1st IEEE Global Conference on Consumer Electronics 2012. IEEE, 2012. P. 120—124.

14. **Hyun W., Huh M. Y., Kim S. H., Kang S. G.** Study on design and implementation of audience measurement functionalities for digital signage service using Kinect camera // 16th International Conference on Advanced Communication Technology. IEEE, 2014. P. 597—600.

15. **Tyukov A., Ushakov A., Shcherbakov M.** et al. Digital signage based building energy management system: solution concept // World Applied Sciences Journal. 2013. Vol. 24. P. 183—190.

16. **Urli S., Blay-Fornarino M., Collet P.** et al. Managing a software ecosystem using a multiple software product line: a case study on digital signage systems // 40th EUROMICRO Conference on Software Engineering and Advanced Applications. IEEE, 2014. P. 344—351.

17. **She J., Crowcroft J.** Smart Signage: A draggable cyber-physical broadcast/multicast media system // IEEE Transactions on Emerging Topics in Computing. 2013. Vol. 1, No. 2. P. 232—243.

18. **Fu H.** Smart signage: a cyber-physical interactive display system for effective advertising. Thesis (M. Phil.) Hong Kong University of Science and Technology, 2013.

19. **Perera C., Liu C. H., Jayawardena S.** The emerging internet of things marketplace from an industrial perspective: A survey // IEEE Transactions on Emerging Topics in Computing. 2015. Vol. 3, No. 4. P. 585—598.

20. **Elhart I., Langheinrich M., Memarovic N.** Integrating interactive applications with digital signage: Towards a scheduling framework for pervasive displays // Pervasive Computing and Communications Workshops (PERCOM Workshops), IEEE International Conference on. IEEE, 2014. P. 495—499.

21. **Савельев А. И.** Оптимизация алгоритмов распределения потоков мультимедийных данных между сервером и клиентом в приложениях видеоконференцсвязи // Труды СПИИРАН. 2013. № 8. С. 61—79.

22. **Saveliev A. I., Vatamaniuk I. V., Ronzhin An. L.** Architecture of data exchange with minimal client-server interaction at multipoint video conferencing // NEW2AN/ruSMART. 2014. LNCS 8638. P. 164—176.

23. **Левоневский Д. К., Ватаманюк И. В., Савельев А. И., Денисов А. В.** Корпоративная информационная система обслуживания пользователей как компонент киберфизического интеллектуального пространства // Известия высших учебных заведений. Приборостроение. 2016. Т. 59, № 11. С. 906—912.

24. **Ронжин А. Л., Юсупов Р. М.** Многомодальные интерфейсы автономных мобильных робототехнических комплексов // Известия Южного федерального университета. Технические науки. 2015. № 1 (162). С. 195—206.

25. **Saveliev A. I., Ronzhin A. L.** Algorithms and Software Tools for Distribution of Multimedia Data Streams in Client Server Videoconferencing Applications // Pattern Recognition and Image Analysis. 2015. Vol. 25, No. 3. P. 517—525.

26. **Ронжин А. Л., Карпов А. А., Лобанов Б. М.** и др. Фонетико-морфологическая разметка речевых корпусов для распознавания и синтеза русской речи // Информационно-управляющие системы. 2006. № 6. С. 24—34.

27. **Ватаманюк И. В., Ронжин А. Л.** Применение методов оценивания размытости цифровых изображений в задаче аудиовизуального мониторинга // Информационно-управляющие системы. 2014. № 4. С. 16—23.

MINOS Multimodal Information and Navigation Cloud System for the Corporate Cyber-Physical Smart Space

D. K. Levonevskiy, DLewonewski.8781@gmail.com,
I. V. Vatamaniuk, vatamaniuk@iias.spb.su, **A. I. Saveliev,** antoni-fox@yandex.ru,
Saint-Petersburg Institute for Informatics and Automation of Russian Academy of Science,
Saint Petersburg, 199178, Russian Federation

Corresponding author:

Levonevskiy Dmitriy K., Researcher, Saint Petersburg Institute for Informatics and Automation of Russian Academy of Science, Saint Petersburg, 199178, Russian Federation
E-mail: DLewonewski.8781@gmail.com

Received on August 03, 2016
Accepted on December 13, 2016

The topic of the article is the architecture of MINOS corporate multimodal information and navigation system, and description of the system components and functioning of a corporate television subsystem. The applied Digital Signage (DS) technology is widely used for transmission of the advertising content and marketing, as well as in education, corporate communications, and multiuser negotiation systems.

The survey presents several examples of DS technologies in information systems for various purposes. The developed infrastructure for operation of the system, including 4 stationary monitors, 4 nettops, network equipment, a web server, and a database server, is also presented. The client and the server software for managing of the corporate TV system software ensure administration and support for the relevant content through the web interface.

Scenarios of using the system for different categories of users, as well as the principles of content management, which is displayed on the screen, based on special signals and events generated by time stamp or by the results of the user interaction, are considered. The primary function of the server is a dynamic formation of the schedules for the media files displayed on the screens. A schedule is an ordered list (queue). New media files are added to the end of the queue, while the media played from its beginning on the screens are excluded from the queue. The corporate TV system broadcasts information to the stationary screens (client systems) in accordance with the schedule, which is updated by the server dynamically based on a number of active events. In the corporate television system, each screen is characterized by an identifier, a description, an access key, a mode of operation and has its own schedule. The screens can be combined into groups and control all the screens in the same group as a single screen. Scenarios of using the system for different categories of users, as well as the principles of content management, which is displayed on screen, based on special signals and the events generated by time stamp or by the results of the user interaction, are considered.

The further area for development is interactivity and personalization of the corporate television system by integrating it with the system of the biometric and RFID identification of the users. For each employee, who is automatically recognized, the corporate TV system will provide the output of the personalized information.

It is expected to implement the management of the information output by means of gestures and speech. Another direction for development of the system is a navigation service, which allows a visitor to determine his (hers) location in the organization by scanning labels with QR-codes. A web application allows the visitor to orientate himself (herself) in the organization, using a laptop computer, and get the necessary background information.

Keywords: *cyber-physical systems, multimodal interfaces, corporate information systems, information and navigation systems, cloud systems*

Acknowledgements: *This work was supported by the Russian Science Foundation, project no. 15-18-00047 and by the Saint Petersburg Institute for Informatics and Automation of Russian Academy of Science (project no. 0073-2015-0005).*

For citation:

Levonevskiy D. K., Vatamaniuk I. V., Saveliev A. I. MINOS Multimodal Information and Navigation Cloud System for the Corporate Cyber-Physical Smart Space, *Programmnaya Ingeneria*, 2017, vol. 8, no 3, pp. 120—128.

DOI: 10.17587/prin.8.120-128

References

1. **Ronzhin An. L., Vatamanyuk I. V., Ronzhin Al. L., Zelezny M.** Matematicheskie metody ocenki razmytosti izobrazhenija i raspoznavanija lic v sisteme avtomaticheskoj registracii uchastnikov soveshhanij (Mathematical Methods to Estimate Image Blur and Recognize Faces in the System of Automatic Conference Participant Registration), *Avtomatika i telemehanika*, 2015, no. 11, pp. 132—144 (in Russian).

2. **Glazkov S. V., Ronzhin A. L.** Kontekstno-zavisimye metody avtomaticheskoj generacii mnogomodal'nyh pol'zovatel'skih veb-interfejsov (Context-dependent methods of automatic generation of multimodal user web-interfaces), *Trudy SPIIRAN*, 2012, no. 2, pp. 170—183 (in Russian).

3. **Ronzhin A. L., Karpov A. A., Leontyeva An. B., Kostuchenko B. E.** Razrabotka mnogomodal'nogo informacionnogo kioska (The development of the multimodal information kiosk), *Trudy SPIIRAN*, 2007, no. 5, pp. 227—245 (in Russian).

4. **Han S., Kim N., Choi K., Kim J.** Design of multi-party meeting system for interactive collaboration, *2nd International Conference on Communication Systems Software and Middleware*. IEEE, 2007, pp. 1—8.

5. **Kelsen K.** *Unleashing the power of digital signage: content strategies for the 5th screen*, CRC Press, 2012.

6. **She J., Crowcroft J., Fu H., Ho P.** Smart signage: An interactive signage system with multiple displays. Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), *IEEE International Conference on and IEEE Cyber, Physical and Social Computing,* IEEE, 2013, pp. 737—742.

7. **Davies N., Clinch S., Alt F.** Pervasive displays: understanding the future of digital signage, *Synthesis Lectures on Mobile and Pervasive Computing*, 2014, vol. 8, no. 1, pp. 1—128.

8. **Ahn S., Song S., Yang J., Choi J.** Web-based multi-screen digital signage, *IEEE 3rd Global Conference on Consumer Electronics (GCCE),* IEEE, 2014, pp. 244—245.

9. **Hossain M. A., Lee Y. T., Lee H., Lyu W., Hong C. H., Nguyen T., Le N. T., Jang Y. M.** A Symbiotic Digital Signage system based on display to display communication, *Seventh International Conference on Ubiquitous and Future Networks,* IEEE, 2015, pp. 124—128.

10. **Hossain M. A., Islam A., Le N. T., Lee Y. T., Lee H. W., Jang Y. M.** Performance Analysis of Smart Digital Signage System Based on software-defined IoT and Invisible Image Sensor Communication, *International Journal of Distributed Sensor Networks*, 2016, vol. 12, no. 7, pp. 1—14.

11 **Kim E., Lee H. J., Lee D. H., Jang U., Kim H. S., Cho K. S., Ryu W.** Efficient contents sharing between digital signage system and mobile terminals, *Advanced Communication Technology (ICACT), 15th International Conference on,* IEEE, 2013, pp. 1002—1005.

12. **Farinella G. M., Farioli G., Battiato S., Leonardi S., Gallo G.** Face re-identification for digital signage applications, *International Workshop on Video Analytics for Audience Measurement in Retail and Digital Signage,* Springer International Publishing, 2014, pp. 40—52.

13. **Lee Y. T., Lien C. H., Hung A., Ren J. H., Chang T. K.** Design of a low cost interactive digital signage, *The 1st IEEE Global Conference on Consumer Electronics,* IEEE, 2012, pp. 120—124.

14. **Hyun W., Huh M. Y., Kim S. H., Kang S. G.** Study on design and implementation of audience measurement functionalities for digital signage service using Kinect camera, *16th International Conference on Advanced Communication Technology,* IEEE, 2014, pp. 597—600.

15. **Tyukov A., Ushakov A., Shcherbakov M., Brebels A., Kamaev V.** Digital signage based building energy management system: solution concept, *World Applied Sciences Journal*, 2013, no. 24, pp. 183—190.

16. **Urli S., Blay-Fornarino M., Collet P., Mosser S., Riveill M.** Managing a software ecosystem using a multiple software product line: a case study on digital signage systems, *40th EUROMICRO Conference on Software Engineering and Advanced Applications,* IEEE, 2014, pp. 344—351.

17. **She J., Crowcroft J.** Smart Signage: A draggable cyber-physical broadcast/multicast media system, *IEEE Transactions on Emerging Topics in Computing*, 2013, vol. 1, no. 2, pp. 232—243.

18. **Fu H.** Smart signage: a cyber-physical interactive display system for effective advertising. Thesis (M. Phil.), Hong Kong University of Science and Technology, 2013.

19. **Perera C., Liu C. H., Jayawardena S.** The emerging internet of things marketplace from an industrial perspective: A survey, *IEEE Transactions on Emerging Topics in Computing*, 2015, vol. 3, no. 4, pp. 585—598.

20. **Elhart I., Langheinrich M., Memarovic N.** Integrating interactive applications with digital signage: Towards a scheduling framework for pervasive displays, *Pervasive Computing and Communications Workshops (PERCOM Workshops), IEEE International Conference on*, IEEE, 2014, pp. 495—499.

21. **Saveliev A. I.** Optimizacija algoritmov raspredelenija potokov mul'timedijnyh dannyh mezhdu serverom i klientom v prilozhenijah videokonferencsvjazi (Optimization algorithms distribution streams of multimedia data between server and client in videoconferencing application), *Trudy SPIIRAN*. 2013, no. 8, pp. 61—79 (in Russian).

22. **Saveliev A. I., Vatamaniuk I. V., Ronzhin An. L.** Architecture of data exchange with minimal client-server interaction at multipoint video conferencing, *NEW2AN/ruSMART,* LNCS 8638, 2014, pp. 164—176.

23. **Levonevskiy D. K., Vatamaniuk I. V., Saveliev A. I., Denisov A. V.** Korporativnaja informacionnaja sistema obsluzhivanija pol'zovatelej kak komponent kiberfizicheskogo intellektual'nogo prostranstva (Corporate information service system as a component of cyber-physical smart space). *Izvestiya vysshikh uchebnykh zavedeniy. Priborostroenie*, 2016, vol. 59, no. 11, pp. 906—912 (in Russian).

24. **Ronzhin A. L., Yusupov R. M.** Mnogomodal'nye interfejsy avtonomnyh mobil'nyh robototehnicheskih kompleksov (Multimodal interfaces of autonomous mobile robots), *Izvestija Juzhnogo federal'nogo universiteta. Tehnicheskie nauki,* 2015, no. 1 (162), pp. 195—206 (in Russian).

25. **Saveliev A. I., Ronzhin A. L.** Algorithms and Software Tools for Distribution of Multimedia Data Streams in Client Server Videoconferencing Applications, *Pattern Recognition and Image Analysis*, 2015, vol. 25, no. 3, pp. 517—525.

26. **Ronzhin A. L., Karpov A. A., Lobanov B. M. , Tsirulnik L. I., Jokisch O.** Fonetiko-morfologicheskaja razmetka rechevyh korpusov dlja raspoznavanija i sinteza russkoj rechi (Phonetic-morphological mapping of speech corpora for recognition and synthesis of Russian speech), *Informacionno-upravljajushhie sistemy*, 2006, no. 6, pp. 24—34 (in Russian).

27. **Vatamaniuk I. V., Ronzhin A. L.** Primenenie metodov ocenivanija razmytosti cifrovyh izobrazhenij v zadache audiovizual'nogo monitoringa (Application of Digital Images Blur Estimation Methods for Audiovisual Monitoring), *Informacionno-upravljajushhie sistemy*, 2014, no. 4, pp. 16—23 (in Russian).

УДК 004.4 DOI: 10.17587/prin.8.129-135

С. А. Леоновец, ст. инженер[1], аспирант[2], e-mail: ser2694@ya.ru,
А. В. Гурьянов[1], ген. директор, e-mail: postmaster@elavt.spb.ru,
А. В. Шукалов, канд. техн. наук, первый зам. ген. директора — гл. конструктор[1], доц.[2], e-mail: aviation78@mail.ru, **И. О. Жаринов**, д-р техн. наук, доц., руководитель уч.-науч. центра[1], зав. кафедрой[2], e-mail: igor_rabota@pisem.net
[1] АО "ОКБ "Электроавтоматика", г. Санкт-Петербург
[2] Университет ИТМО, г. Санкт-Петербург

Программное обеспечение для автоматизации подготовки текстовой конструкторской документации на программно-управляемые изделия

Рассмотрена задача подготовки текстовой конструкторской документации автоматизированным способом с помощью специализированного программного обеспечения. Автоматизация процесса подготовки такой документации основана на специализированной обработке инженерных данных, заданных в нормативно-технической документации или в техническом задании. Обработка инженерных данных предполагает математический анализ технических характеристик изделия и семантический анализ тактических требований к изделию. Технические характеристики и тактические требования приводятся в структурированных электронных документах, подготавливаемых в распространенных файловых форматах в соответствии с шаблонами, разработанными на основе отраслевых стандартов. В форме графа представлены номенклатура разрабатываемых конструкторских документов и пример обработки данных для генерации текстового документа. Описано разработанное программное обеспечение и инструментальные средства, доступные разработчику в проектной деятельности на приборостроительном предприятии.

Ключевые слова: *конструкторская документация, автоматизация, текстовые документы, генерация*

Введение

Процесс разработки программно-управляемых изделий сопровождается подготовкой конструкторской документации (КД) на различных этапах проектирования, таких как эскизное проектирование, технический проект, технические предложения, рабочее конструкторское проектирование. Номенклатура разрабатываемых документов регламентируется соответствующими государственными стандартами, в частности, ГОСТ 2.102-68, и зависит от сложности изделия (системы, комплекса) и его практической области применения.

В распоряжении разработчиков проектных организаций на настоящее время имеются различные программные средства, автоматизирующие процесс подготовки КД. Широкое практическое применение получили системы автоматизации проектирования (САПР) [1—4], такие как AutoCAD, Solidworks, UniGraphics, предназначенные для оформления чертежной документации; Cadence Design, Mentor Graphics, XILINX Foundation, ALTERA, предназначенные для подготовки схемной документации и данных проекта; а также текстовый процессор Word, используемый для оформления текстовой документации и т. д.

Основной особенностью применения таких САПР является их узкая специализация, обеспечивающая разработчику возможность автоматизировать процесс подготовки ограниченного числа видов КД в отрыве от общего числа документов, разрабатываемых на проект в целом. Таким образом, разработчики вынуждены устанавливать на инструментальную ЭВМ автоматизированного рабочего места (АРМ) различные виды САПР и оформлять техническую документацию проекта поочередно — сначала одну группу КД с использованием одного вида САПР, затем другую группу КД с использованием другого вида САПР и т. д. При этом эффект автоматизации процесса разработки КД от применения САПР резко снижается и в ряде случаев взаимосвязь между инженерными данными, являющимися составной частью различных документов, оказывается сосредоточена только в памяти разработчика, что создает предпосылки для возникновения ошибок проектирования.

В связи с этим актуальной является задача разработки отраслевых САПР, позволяющих разработчикам осуществлять автоматизированную поддержку процесса сквозного проектирования КД на изделие. Принципы построения отраслевой САПР

в области авиационного приборостроения подробно рассмотрены в работах [5—10]. Цель настоящей статьи заключается в представлении широкому кругу читателей результатов выполнения составной части опытно-конструкторской работы по созданию в АО "ОКБ "Электроавтоматика" специализированного программного обеспечения. Оно представляет собой отдельный компонент отраслевой САПР для автоматизированного формирования взаимосвязанных текстовых конструкторских документов на программно-управляемые изделия.

1. Номенклатура разрабатываемых конструкторских документов

Основополагающим общетехническим документом, регулирующим на предприятии отношения заказчика (военного представителя) и разработчика в части объема и видов конструкторских документов, разрабатываемых на программно-управляемое изделие на определенном этапе проектирования, является протокол номенклатуры КД. В нем указывают:

- исчерпывающий перечень КД, подготавливаемой на изделие и на его сборочные единицы;
- используемые при этом САПР, включая версии программного обеспечения, установленные на инструментальной ЭВМ АРМ;
- подразделения — ответственных исполнителей по разработке и согласованию (метрология, нормоконтроль, технологический контроль и т. д.) КД;
- способ (электронный, на бумажных носителях) и сроки хранения документации в архиве предприятия.

На рис.1 приведен граф КД (пример), разрабатываемой на программно-управляемое изделие. Стрелками схематично показаны взаимосвязи между документами — инженерные данные, являющиеся исходными для подготовки пошагово разрабатываемых видов конструкторских документов.

На рис. 1 используются обозначения (аббревиатуры), принятые в отечественной промышленности для различных кодов конструкторских документов: УЧ — упаковочный чертеж; ГЧ — габаритный чертеж; МЧ — монтажный чертеж; МЭ — электромонтажный чертеж; ПФ — патентный формуляр; ТУ — технические условия; РПП — руководство по применению и программированию (общетехнический документ); ВО — чертеж общего вида; СБ — сборочный чертеж; Т10М — данные о результатах проектирования платы; Д12 — данные контроля; Д11 — данные программирования; Д13 — данные проекта; ФПО — функциональное программное обеспечение; ТЗ — техническое задание; Е1 — схема деления изделия на составные части; Э1 — схема электрическая структурная; Э2 — схема электрическая функциональная; Э3 — схема электрическая принципиальная; Э4 — схема электрическая соединений; Э5 — схема электрическая подключения; И5 — инструкция по занесению данных и контролю; ПС1 — Паспорт. Приложение. Контрольные суммы; ЭД — ведомость эксплуатационных документов; ОП — опись папки; ВС — ведомость спецификаций;

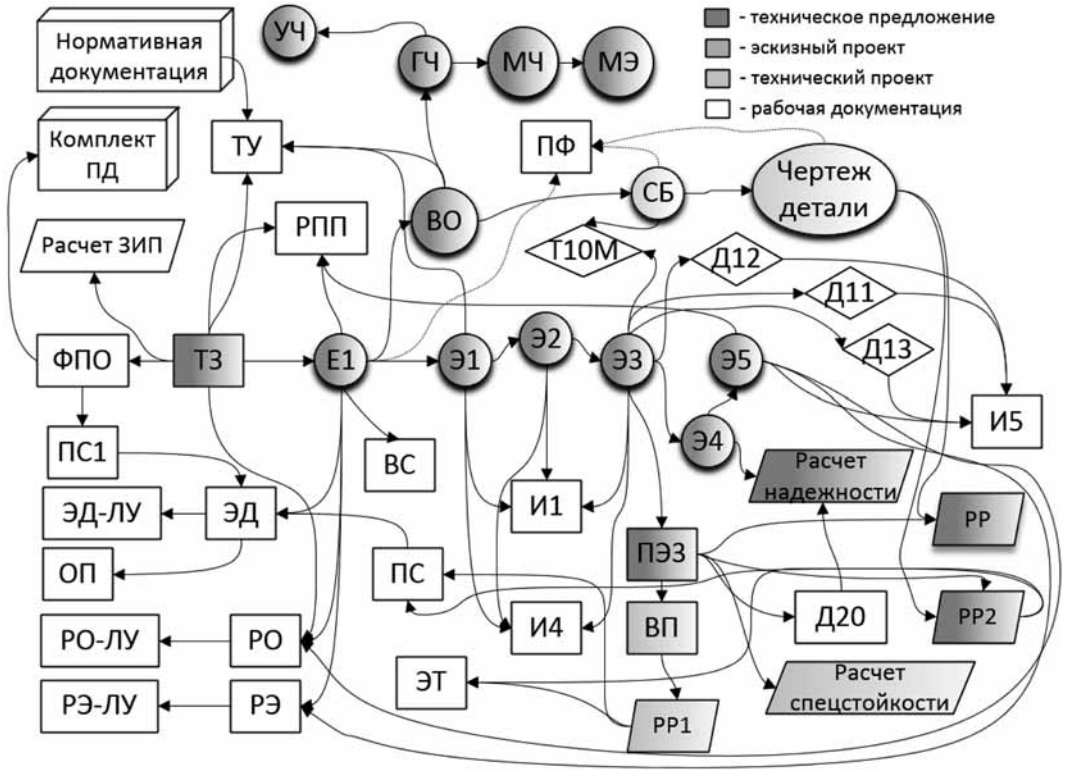

Рис. 1. Граф КД, разрабатываемой на программно-управляемое изделие (пример)

И1 — инструкция по проверке, настройке, регулировке; И4 — инструкция по проверке; ПЭ3 — перечень элементов; ВП — ведомость покупных изделий; РР1 — расчет драгоценных металлов; РР2 — расчет цветных металлов; Д20 — карта рабочих режимов; РР — расчет показателей уровня стандартизации и унификации; ПС — паспорт; РО — руководство по обслуживанию; РЭ — руководство по технической эксплуатации; ЭТ — этикетка; ЛУ — лист утверждения; ЗИП — запасные изделия прилагаемые.

Полутонами (в оттенках серого цвета) на рис. 1 размечены элементы графа, соответствующие конструкторским документам, разрабатываемым на различных этапах проектирования изделия.

2. Описание инструментального средства проектирования взаимосвязанных конструкторских документов

Инструментальное средство проектирования взаимосвязанных конструкторских документов представляет собой программный продукт, интерфейс пользователя которого показан на рис. 2, см. вторую сторону обложки. В рабочем окне программы отображается дерево разрабатываемого проекта *1*, рабочая область окна предварительного просмотра *2*, рабочая область окна редактирования *3*, меню разработчика *4*, панель инструментов разработчика *5* и строка состояния процесса проектирования *6*.

Дерево разрабатываемого проекта представляет собой удобный для навигации по проекту программный инструмент, в котором отображаются нормативно-техническая документация, необходимая для генерации конструкторской документации, документы служебной переписки разработчика и заказчика, а также разрабатываемые конструкторские документы.

Перечень разрабатываемых конструкторских документов сопровождается в рабочем окне программы указанием на их версионную принадлежность для каждого документа и состоянием разработки. В процессе разработки каждый документ может принимать следующие состояния: "черновик" (*draft*), "проверка" (*proposed*), "отклонен" (*declined*), "принят" (*approved*). Пример перехода атрибута конструкторского документа из одного состояния разработки в другое представлен на рис. 3.

Дерево проекта унаследовано от API-функции (*Application Programming Interface*) *System.Windows.Controls.TreeView* операционной системы Windows инструментальной ЭВМ АРМ. Каждый вид элемента разрабатываемого конструкторского документа представлен соответствующим видом ярлыка в рабочем окне программы в поле *1* (см. рис. 2, вторая сторона обложки).

Рабочая область окна предварительного просмотра *2* (см. рис. 2, вторая сторона обложки) позволяет разработчику оценивать содержание и оформление конструкторского документа при его автоматизированной генерации. Инструментальные средства САПР позволяют разработчику просматривать проект документа (режим предварительного просмотра)

Рис. 3. Пример изменения атрибута конструкторского документа в процессе разработки КД

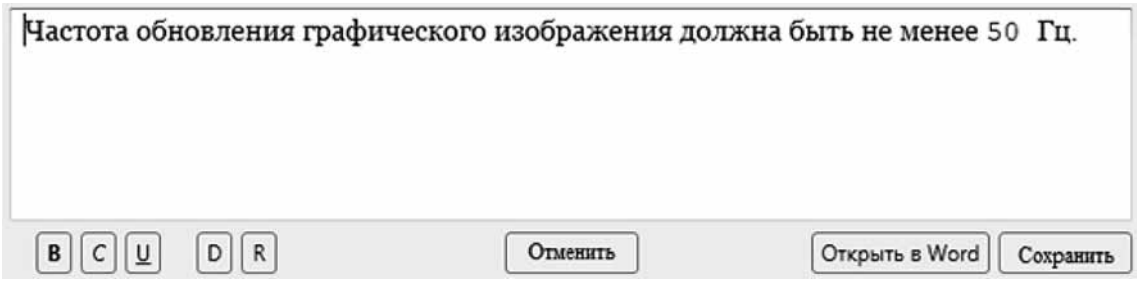

Рис. 4. Внешний вид окна редактора текстовой информации, содержащейся в конструкторском документе

перед его сохранением на диске и, при необходимости, распечатать его на бумажный носитель. В распоряжении разработчика следующие функции:

— функция поиска текста в конструкторском документе по ключевым словам, заданным в строке поиска;

— функция масштабирования документа для удобства его предварительного просмотра.

Рабочая область окна редактирования *3* (см. рис. 2, вторая сторона обложки) реализована для внесения изменений в текст разрабатываемого документа. Окно редактора текста, доступное разработчику, представлено на рис. 4. Обеспечиваются возможности изменения шрифта текста, полужирного или курсивного начертания текста, подчеркивания фрагментов текста. Также в редакторе предусмотрена возможность использования фрагментов текста, заимствованных из буфера обмена. Связь инженерных данных между конструкторскими документами основана на структурировании текстов, содержащих требования или определения, имеющиеся в первичных нормативных документах или в конструкторских документах, разработанных ранее.

Меню разработчика *4* (см. рис. 2, вторая сторона обложки) в рабочем окне программы представлено перечисленными далее опциями.

- "Проект" — включает стандартные функции для работы с текущим проектом (создание нового проекта, загрузка существующего проекта из дискового накопителя, сохранение внесенных изменений в проекте). Опция позволяет выбрать шаблон документа, созданный на предприятии на основе требований отраслевых стандартов, а также сохранять документ на диске инструментальной ЭВМ АРМ в универсальном формате XML (eXtensible Markup Language) для поддержки просмотра и редактирования документа в стандартных текстовых редакторах.
- "Конфигурация" — включает функции для создания разработчиком конфигурации атрибутов элементов конструкторского документа, зависящих от типа документа и соответствующей нормативно-технической документации. Атрибуты документыов могут быть одного из следующих типов: произвольный текст; список из нескольких элементов, который формирует разработчик конфигурации; логическое значение.
- "Инструменты" — включает функции автоматизированного контроля соответствия документа требованиям нормативно-технической документации, а также функции генерации документа в формате текстового процессора Word.
- "Настройки" — подменю, где разработчику доступно изменение настроек функционирования рабочей программы, а именно: частота автосохранения проекта на диске инструментальной ЭВМ, язык интерфейса программы и т. д.
- "Поддержка" — включает подменю с формой для обращения разработчика в службу технической поддержки, организованную разработчиком САПР (при необходимости).
- "О программе" — содержит сведения о версионной принадлежности САПР.

Панели инструментов *5* (см. рис. 2, вторая сторона обложки) разработчика позволяют разработчику получить быстрый доступ к наиболее часто используемым инструментальным средствам программы создания взаимосвязанных конструкторских документов.

Строка состояния *6* (см. рис. 2, вторая сторона обложки) процесса проектирования представляет разработчику справочные данные и допускает следующие виды состояний в работе программы: *Ready* (программа готова к использованию); *Loading* (загрузка проекта или документа); *Generation* (генерация конструкторского документа); *Save* (сохранение проекта на диске).

3. Пример типов данных, используемых для представления инженерных данных в конструкторской документации

Программа разработки текстовой конструкторской документации использует инженерные данные, первично приведенные в составе технического задания на разработку изделия и в нормативно-технической документации (отраслевые стандарты). Исходные документы должны быть подготовлены в электронном виде (файл). Для создания проекта за основу берут тактико-технические требования к разрабатываемому программно-управляемому изделию. Разработчику доступны файлы исходных данных — технического задания и отраслевых стандартов — в распространенных форматах pdf, doc или docx.

Программа разработки КД автоматически добавит в новый или в существующий проект список требований из нормативных документов, определив их по закладкам или содержанию документа. Ниже приведен фрагмент кода программы, написанной на высокоуровневом языке программирования C#, который обрабатывает документ в формате pdf, содержащий тактико-технические требования на изделие.

```
public static void searchRequirement(IList<Dictionary<string, object>> pdfelement,
TreeViewItem mainItem) {
        foreach (Dictionary<string, object> dc in pdfdocument)
  foreach (KeyValuePair<string, object> kvp in dc) {
    // поиск заголовков
    if (kvp.Key == "Title") {
     string s1 = kvp.Value.ToString();
     // очистка лишних символов
     if (Regex.IsMatch(s1[s1.Length - 1].ToString(), "[\\S]") == false)
      s1 = s1.Remove(s1.Length - 1);
     // создание элемента дерева с заголовком требования
     TreeViewItem itemRequirement = new TreeViewItem();
     itemRequirement.Header = s1;
     mainItem.Items.Add(itemRequirement);
    }
    // поиск подзаголовков
    if (kvp.Key == "Kids") {
          IList<Dictionary<string, object>> dict = (IList<Dictionary<string, object>>)kvp.Value;
     // рекурсивный вызов метода
     searchRequirement (dict, itemRequirement);
}
}
}
```

Пусть при этом внутренняя структура конструкторского документа в формате pdf, содержащего инженерные данные на программно-управляемое изделие, имеет следующий вид:

```
<</Title(3.2.2.1 IMAGE CONTRAST)/Parent 14 0 R/First 16 0 R/Last 16 0 R/Next 17 0 R/Dest[5 0
R/FitH 788]/Count 1>>
endobj
17 0 obj
<</Title(3.2.2.2 IMAGE BRIGHTNESS)/Parent 14 0 R/Prev 15 0 R/Dest[5 0 R/FitH 752]>>
endobj
8 0 obj
<</Title(Bookmark)/Parent 7 0 R/First 9 0 R/Last 10 0 R/Next 14 0 R/Dest[1 0 R/FitH 806]/Count 5>>
endobj
14 0 obj
<</Title(3.2.2 VISUAL PARAMETERS)/Parent 7 0 R/First 15 0 R/Last 17 0 R/Prev 8 0 R/Dest[5 0 R/FitH
806]/Count 3>>
endobj
7 0 obj
<</Type/Outlines/First 8 0 R/Last 14 0 R/Count 10>>
endobj
2 0 obj
<</Type/Font/Subtype/Type1/BaseFont/Helvetica/Encoding/WinAnsiEncoding>>
endobj
4 0 obj
<</Type/Pages/Count 2/Kids[1 0 R 5 0 R]/ITXT(5.0.5)>>
endobj
18 0 obj
<</Type/Catalog/Pages 4 0 R/Outlines 7 0 R>>
endobj
```

После обработки программой подготовки конструкторских документов представленный в формате pdf фрагмент текста приводится к структуре файла в формате XAML (*eXtensible Application Markup Language*):

```
<TreeViewItem Header = "3.2.2 VISUAL PARAMETERS">
    <TreeViewItem Header = "3.2.2.1 IMAGE CONTRAST"/>
    <TreeViewItem Header = "3.2.2.2 IMAGE BRIGHTNESS"/>
</TreeViewItem>
```

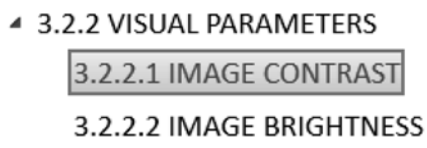

Рис. 5. Визуальное представление разработчику списка требований к изделию в рабочем окне программы подготовки КД (пример)

В программе САПР код XAML представляется в форме иерархического дерева, пригодного для визуального восприятия разработчиком. Область окна рабочей программы проектирования, соответствующая обработанному фрагменту текста в формате pdf, имеет вид, показанный на рис. 5. Список (*Visual parameters*) в рассматриваемом примере включает набор требований к программно-управляемому изделию — средству отображения информации и состоит из двух пунктов: контрастность изображения (*Image contrast*) и яркость изображения (*Image brightness*) на экране.

Таким образом, в программном обеспечении САПР реализована автоматизированная обработка структурированных электронных текстовых конструкторских документов в целях извлечения из них инженерных данных, используемых в конструкторских документах, разрабатываемых на последующих этапах проектирования. Инженерные данные помещаются в новых разрабатываемых документах в соответствующие поля в шаблонах, подготовленных на приборостроительном предприятии на основе отраслевых стандартов оформления отдельных видов текстовых конструкторских документов: перечень элементов, расчетные работы (расчет драгоценных металлов, расчет цветных металлов), ведомость покупных изделий, паспорт, этикетка, патентный формуляр и т. д.

Заключение

Разработка системы автоматизации оформления технической документации является экономически выгодной. Предлагаемое программное обеспечение САПР позволяет автоматизировать на приборостроительном предприятии работу разработчиков программно-управляемых изделий, связанную с подготовкой технической текстовой документации.

Программа САПР имеет объективные преимущества по отношению к известным программным продуктам аналогичного назначения, таким как "3SL Cradle 7.0" и "ДС БАРС КТ-178В", за счет наличия встроенной опции контроля версий, позволяющей разработчику сопровождать изделия, разработанные по разным версиям КД, в жизненном цикле продукции и создавать архивные версии комплектов КД в электронном виде.

С помощью предлагаемого программного обеспечения САПР разработчику уже сегодня доступна функция генерации автоматизированным способом [11] нескольких видов текстовых документов (пояснительная записка, ТУ, РПП) на программно-управляемые изделия за счет наличия соответствующих шаблонов, основанных на отечественных отраслевых стандартах.

Апробация предложенного программного обеспечения САПР проводилась в АО "ОКБ "Электроавтоматика" с использованием действующих в области авиационного приборостроения отечественных стандартов и стандартов ARINC (*Aeronautical Radio Inc.*, США), в частности, ARINC 651-655, при подготовке документации на бортовые цифровые вычислительные системы.

Автоматизация процесса подготовки текстовой конструкторской документации с использованием разработанного программного обеспечения САПР позволила сократить трудоемкость (до 80 % по методике, описанной в работе [12]) и время проектирования мультипроцессорного вычислителя "Крейт", предназначенного для установки на объекты гражданской авиации.

Описанное программное обеспечение САПР разработано на языке программирования C# и функционирует на базе инструментальной ЭВМ со следующими характеристиками: ASUS K56CB-X0391H, процессор Intel(R) Core(TM) i5-3337U, 4 ядра, тактовая частота 1,8 ГГц, оперативная память 6 Гбайт под управлением операционной системы Windows 8.1.

Список литературы

1. **Авдеева М., Чиркин А.** Перевод бумажной документации в электронный вид // САПР и графика. 2004. № 1. С. 70—72.

2. **Садовников Д., Ноздрин А., Ширяев Н.** Система управления технической и проектно-конструкторской документацией // САПР и графика. 2002. № 5. С. 74—77.

3. **Кукаренко Е., Молочко Д**. Управление потоками знаний в техническом документообороте предприятия // САПР и графика. 2001. № 10. С. 35—37.

4. **Бычков И., Ващук Ю.** Конструкторская спецификация — информационная основа управления предприятием // САПР и графика. 2001. № 9. С. 90—95.

5. **Брахутин А. Г.** CALS выходит на федеральный уровень // Вестник авиации и космонавтики. 2001. № 5. С. 26—27.

6. **Гатчин Ю. А., Жаринов И. О., Жаринов О. О.** Архитектура программного обеспечения автоматизированного рабочего места разработчика бортового авиационного оборудования // Научно-технический вестник информационных технологий, механики и оптики. 2012. № 2. С. 140—141.

7. **Гатчин И. Ю., Жаринов И. О., Жаринов О. О., Косенков П. А.** Реализация жизненного цикла "проектирование—производство—эксплуатация" бортового оборудования на предприятиях авиационной промышленности // Научно-технический вестник информационных технологий, механики и оптики. 2012. № 2. С. 141—143.

8. **Парамонов П. П., Гатчин Ю. А., Жаринов И. О.** и др. Принципы построения отраслевой системы автоматизированного проектирования в авиационном приборостроении // Научно-технический вестник информационных технологий, механики и оптики. 2012. № 6. С. 111-117.

9. **Utkin S. B., Batova S. V., Blagonravov S. A., Konovalov P. V., Zharinov I. O.** Automated construction of software configuration tables for real-time systems in avionics // Programming and Computer Software. 2015. Vol. 41, N. 4. P. 219—223.

10. **Благонравов С. А., Уткин С. Б., Батова С. В., Коновалов П. В.** Опыт применения технологии эмуляции процессов при разработке компонентов программного обеспечения авиационных систем // Программная инженерия. 2015. № 8. С. 18—25.

11. **Шек-Иовсепянц Р. А., Жаринов И. О.** Генерация проектных решений бортового оборудования с использованием аппарата генетических алгоритмов // Научно-технический вестник информационных технологий, механики и оптики. 2010. № 3. С. 67—70.

12. **Жаринов И. О., Жаринов О. О., Шек-Иовсепянц Р. А., Суслов В. Д.** Оценка снижения трудоемкости подготовки конструкторской документации с использованием CALS-технологии в приборостроении // Научно-технический вестник информационных технологий, механики и оптики. 2012. № 4. С. 151—153.

Software for Automating the Process of Preparing the Text Design Documentation for the Program-Driven Products

S. A. Leonovets[1,2], e-mail: ser2694@ya.ru, **A. V. Gurjanov**[1], e-mail: postmaster@elavt.spb.ru,
A. V. Shukalov[1,2], e-mail: aviation78@mail.ru, **I. O. Zharinov**[1,2], e-mail: igor_rabota@pisem.net
[1] Design Bureau "Electroavtomatika", Saint Petersburg, 198095, Russian Federation,
[2] Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University), Saint Petersburg, 197101, Russian Federation

Corresponding author:

Zharinov Igor O., Chef of Department, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University), Saint Petersburg, 197101, Russian Federation
E-mail: igor_rabota@pisem.net

Received on November 18, 2016
Accepted on December 06, 2016

The task of automatically preparing the text design documentation by means of a specialized software is considered. Automation of process of preparation of documentation is based on processing the engineering data, given in the specifications and technical documentation or in the specification. Processing of engineering data assumes the analysis of the structured electronic documents prepared in widespread formats according to templates on the basis of industry standards and generating the text design document by an automated method. The nomenclature of the developed design documents and an example of data handling for generation of the text document are provided in the form of a graph. The new developed software and the work benches available to the developer in project activities are described.

Keywords: *design documentation, automation, text document, generation*

For citation:

Leonovets S. A., Gurjanov A. V., Shukalov A. V., Zharinov I. O. Software for Automating the Process of Preparing the Text Design Documentation for the Program-Driven Products, *Programmnaya Ingeneria,* 2017, vol. 8, no. 3, pp. 129—135.

DOI: 10.17587/prin.8.129-135

References

1. **Avdeeva M., Chirkin A.** Perevod bumazhnoj dokumentacii v jelektronnyj vid (Convert paper documents into electronic form), *SAPR i grafika*, 2004, no. 1, pp. 70—72 (in Russian).
2. **Sadovnikov D., Nozdrin A., Shirjaev N.** Sistema upravlenija tehnicheskoj i proektno-konstruktorskoj dokumentaciej (The control system of technical and design documentation), *SAPR i grafika*, 2002, no. 5, pp. 74—77 (in Russian).
3. **Kukarenko E., Molochko D.** Upravlenie potokami znanij v tehnicheskom dokumentooborote predprijatija (Flow management knowledge in the technical document of the enterprise), *SAPR i grafika*, 2001, no. 10, pp. 35—37 (in Russian).
4. **Bychkov I., Vashhuk Ju.** Konstruktorskaja specifikacija—informacionnaja osnova upravlenija predprijatiem (The design specification — the basis of enterprise management information), *SAPR i grafika*, 2001, no. 9, pp. 90—95 (in Russian).
5. **Brahutin A. G.** CALS vyhodit na federal'nyj uroven' (CALS goes to the federal level), *Vestnik aviacii i kosmonavtiki*, 2001, no. 5, pp. 26—27 (in Russian).
6. **Gatchin Yu. A., Zharinov I. O., Zharinov O. O.** Arhitektura programmnogo obespechenija avtomatizirovannogo rabochego mesta razrabotchika bortovogo aviacionnogo oborudovanija (Software Architecture for the Automated Workplace of the Onboard Aviation Equipment Developer), *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2012, no. 2, pp. 140—141 (in Russian).
7. **Gatchin I. Yu., Zharinov I. O., Zharinov O. O., Kosenkov P. A.** Realizacija zhiznennogo cikla "proektirovanie—proizvodstvo—jekspluatacija" bortovogo oborudovanija na predprijatijah aviacionnoj promyshlennosti (The implementation of the life cycle "design—production—operate" on-board equipment to the aviation industry enterprises), *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2012, no. 2, pp. 141—143 (in Russian).
8. **Paramonov P. P., Gatchin Yu. A., Zharinov I. O., Zharinov O. O., Deiko M. S.** Principy postroenija otraslevoj sistemy avtomatizirovannogo pro-ektirovanija v aviacionnom priborostroenii (Principles of Branch System Creation for the Automated Design in Aviation Instrumentation), *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2012, no. 6, pp.111—117 (in Russian).
9. **Utkin S. B., Batova S. V., Blagonravov S. A., Konovalov P. V., Zharinov I. O.** Automated construction of software configuration tables for real-time systems in avionics, *Programming and Computer Software*, 2015, vol. 41, no. 4, pp. 219—223.
10. **Blagonravov S. A., Utkin S. B., Batova S. V., Konovalov P. V.** Opyt primenenija tehnologii jemuljacii processov pri razrabotke komponentov programmnogo obespechenija aviacionnyh sistem (Using Emulation Technics in the Development of Avionics Software Components), *Programmnaya Ingeneria*, 2015, no. 8, pp. 18—25 (in Russian).
11. **Shek-Iovsepjanc R. A., Zharinov I. O.** Generacija proektnyh reshenij bortovogo oborudovanija s ispol'zovaniem apparata geneticheskih algoritmov (Design Generation of the Avionic Equipment by Genetic Algorithms), *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2010, no. 3, pp. 67—70 (in Russian).
12. **Zharinov I. O., Zharinov O. O., Shek-Iovsepjanc R. A., Suslov V. D.** Ocenka snizhenija trudoemkosti podgotovki konstruktorskoj dokumentacii s ispol'zovaniem CALS-tehnologii v priborostroenii (Assessment of reducing the complexity of the preparation of the design documentation using the CALS-technologies in instrument), *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2012, no. 4, pp. 151—153 (in Russian).

УДК 519.688, 669.14.242 DOI: 10.17587/prin.8.136-139

Н. Н. Светушков, доц., e-mail: svt.n.n@mail.ru, Московский авиационный институт (Национальный исследовательский университет)

Метаязык описания сложных 3D-объектов для прикладных информационных систем

Предложен подход к описанию сложных трехмерных объектов с помощью специализированного метаязыка. Основные положения такого языка описания связей разработаны и реализованы на C++. Показана его эффективность при разработке прикладных информационных систем, требующих графической иллюстрации реальных изделий. Приведены примеры использования метаязыка и продемонстрирована разработанная с его помощью трехмерная модель космического аппарата "Спектр-РГ", которая применяется в информационной системе по обработке телеметрической информации. Система классов, поддерживающая данный метаязык, позволяет программисту-разработчику создавать собственные трехмерные объекты в приложениях, что значительно повышает коммерческую привлекательность создаваемых программных средств.

Ключевые слова: *3D-графика, язык для создания объектов, программные средства, информационные системы*

Введение

Наиболее удобным способом для анализа состояния технического изделия или конструкции является визуальное представление в виде трехмерной модели (3D-модели), которая дополняется необходимыми программными ресурсами и данными. Создание таких программно-информационных систем особенно актуально для задач, возникающих при анализе поведения удаленных на большие расстояния космических аппаратов (КА), что в свою очередь связано с обработкой большого объема телеметрической информации, отражающей функциональное состояние аппарата.

В настоящее время существует большое число программных систем, предназначенных для формирования разнообразных трехмерных моделей, причем их число продолжает увеличиваться. Например, к одной из наиболее распространенных на промышленных предприятиях систем подобного назначения относится программный комплекс Solid Works, позволяющий создавать сложные структурированные объекты. Как и в большинстве аналогичных программных средств, в нем реализован подход, при котором сначала выполняется построение элементарной фигуры (параллелепипед, шар, конус, цилиндр и т. д.), а затем ее модификация путем различных трансформаций — вытягивание граней в определенном направлении, объединение, "вычитание" и др. Однако в силу большого числа используемых операций, эти программные средства, как правило, имеют сложный иерархический интерфейс. Для его изучения обычно разрабатывают специальные руководства по обучению пользователя основам проектирования (например, для 3D Max или Blender 3D). Кроме того, завышенные требования к таким средствам по объему памяти и по быстродействию не позволяют устанавливать их на планшеты, работающие под управлением операционной системы Android. А такие планшеты в настоящее время получили широкое распространение.

Главным недостатком существующих в области 3D-моделирования коммерческих программных систем является их закрытый характер. Для отображения построенной с их помощью трехмерной модели требуется дополнительно установить специализированную программную среду ("вьюер"). Закрытый характер системы не позволяет разработчику использовать созданные трехмерные объекты в собственных приложениях. Это ограничение, как правило, приводит к отсутствию возможности обеспечить наглядное графическое представление обрабатываемой информации в прикладном программном обеспечении, и в конечном счете снижает эффективность и коммерческую привлекательность разрабатываемых программных средств.

В настоящей работе описаны базовые (концептуальные) положения разрабатываемого метаязыка создания трехмерных объектов для упрощенного графического редактора без завышенных требований к памяти и быстродействию компьютера, что позволит использовать его для работы в операционных системах типа Android. Обеспечение функциональности метаязыка осуществляется путем подключения разработанной автором связанной системы классов, обеспечивающих перевод описательных лексем в графическое изображение. Самым главным преимуществом разрабатываемой системы классов является ее открытый характер, что дает возможность программисту-разработчику использовать созданные с помощью метаязыка объекты в любых программах, написанных на языке C++ (при условии дополнительного подключения свободно распространяемой библиотеки DirectX).

Отметим, что разработчики Microsoft использовали аналогичный подход в офисном приложении Word,

когда была предложена концепция языка XML. На основе лексем этого языка пользователь имеет возможность создавать любые двумерные картинки в текстовом файле "на лету", не пользуясь дополнительными программами редактирования изображений (типа PowerPoint).

Базовые положения метаязыка описания связей

Предлагаемые автором базовые положения для построения трехмерных моделей, которые в дальнейшем были использованы при создании модели КА "Спектр-РГ", основаны на определении в качестве базового графического элемента не точки, а грани (рис. 1, см. третью сторону обложки). При таком подходе для идентификации любой точки в пространстве необходимо указывать не абсолютные трехмерные координаты в какой-то выделенной системе координат, а локальные, связанные с выбранной гранью модели. Такой подход приводит к появлению нового качества проектируемого графического объекта — гибкости трехмерной модели. Таким образом, метаязык описания связей, на котором основан метод построения трехмерных моделей, предоставляет пользователю возможность произвольным образом видоизменять отдельные части модели, не заботясь о ее целостности, которая поддерживается программным способом именно за счет установленных связей.

Локальная система координат, связанная с выбранной гранью, определяется ограничивающими ребрами и позволяет идентифицировать любую точку путем задания двух ее координат на грани или трех координат в пространстве.

Таким образом, любая точка на поверхности готового изделия может быть описана тремя параметрами — номером входящей в него грани и двумя плоскими координатами. Отметим, что в этом подходе координатные оси на плоскости генерируются автоматически и не обязательно должны быть ортогональны, что делает общую структуру модели более гибкой.

Для создания составных трехмерных моделей были определены базовые классы, которые обеспечивают генерацию элементарных геометрических фигур, таких как параллелепипед, пирамида, шар, цилиндр и др. Как уже отмечалось ранее, несмотря на то что эти фигуры присутствуют во всех средствах разработки, предлагаемые приемы их формирования в данном случае отличаются от общепринятых. Отметим, что все разработанные классы, в которых и формируются упомянутые выше классические фигуры, содержат в качестве основного элемента грань. Общая иерархия классов, созданных для формирования элементарных объектов, устроена таким образом, чтобы для каждого элементарного объекта была возможность определить точки контакта (связи) с другим произвольным объектом, входящим в состав этой структуры. Таким образом, методы класса включают в себя общие средства по доступу к контактным плоскостям и точкам на них. Как следствие, эти методы формируют замкнутый взаимосвязанный набор классов, позволяющий определять произвольные связи между объектами и по ним строить новые объекты.

Созданный метаязык включает в себя следующие команды: добавление в сцену элементарных объектов в любом месте программы; доступ ко всем элементарным объектам через текущие указатели; установку контактных поверхностей между объектами из разных классов; перемещение объектов в пространстве по заданным направлениям; изменение размеров объекта и его ориентации в пространстве; задание свойств цвета и прозрачности, а также ряд других.

Дополнительным немаловажным преимуществом разработанного языка является возможность иерархической группировки элементарных объектов, что позволяет воспроизводить сложные вложенные структуры, к которым относится, например, модель КА "Спектр-РГ".

Использование метаязыка при разработке прикладного программного обеспечения

Предложенная структура метаязыка была реализована в среде разработки Visual C++ с подключенными библиотеками DirectX [1—4], позволяющими выводить создаваемые трехмерные изображения сначала в видеобуфер, а потом на экран. При создании объекта, показанного на рис. 2 (см. третью сторону обложки), были использованы команды присоединения параллелепипедов и цилиндров к выделенным граням, а также возможность задавать некоторые объекты невидимыми. Отметим, что возможность задавать фигуру невидимой является очень удобным программистским приемом, позволяющим определенным образом группировать элементарные объекты в общую связанную структуру.

Рассмотрим пример создания трехмерного изделия путем соединения двух плоскостей с помощью цилиндра заданного диаметра. Для этого используется метод *Contact,* входящий в методы класса *Tube.* Этот метод позволяет сформировать цилиндр (трубку), соединяющий любые две грани (плоскости), которые могут принадлежать различным фигурам.

В результате выполнения этой команды формируется трубка, которая заключена между двумя плоскостями (рис. 2, см. третью сторону обложки). Длина трубки и координаты точек нижней и верхней граней вычисляются автоматически, а ее диаметр определяется ранее методом *Shape* класса *Tube.*

Как отмечалось выше, каждая грань имеет свой базис, и поэтому к любой точке на грани можно осуществить доступ путем задания двух координат. Для того чтобы создать полый цилиндр, соединяющий две точки на двух различных плоскостях, в указанный выше метод передаются указатели на каждую из плоскостей и задаются соответственно четыре параметра — по два каждой из плоскостей. Координаты задаются в безразмерном виде, относительно длины каждого из базисных векторов. Первый указатель на плоскость — *Main.Plane(iMain_Plane),* второй — *Main.Plane(iMain_Plane + 1)).* Последующие две пары значений типа float и есть координаты, определяющие положение трубки относительно поверхностей.

Таким образом, использованием одного метода не только формируется объект в виде трубки, но и автоматически вычисляется его длина и определяется его

пространственная ориентация. Использование методов такого типа позволяет быстро создавать сложные геометрические объекты, такие как, например, топливный бак с дополнительным оборудованием (рис. 3, см. третью сторону обложки).

На рис. 4 (см. четвертую сторону обложки) представлена трехмерная модель несущей фермы, на которой закреплено порядка 50 температурных датчиков, выполняющих контроль температурных режимов КА. Вследствие большого числа датчиков и необходимости их точного позиционирования на несущей ферме, моделирование этого узла представляло довольно трудоемкую задачу. Присоединение датчиков к цилиндрам выполнялось командой *ATTACH*, в которой задавался номер элементарной грани цилиндра, координаты точки контакта на этой грани и высота параллелепипеда (в этой команде предполагается построение параллелепипеда по внешней нормали к контактной грани). Соединение цилиндров к основной поверхности выполнялось командой *CONTACT*, в которой автоматически рассчитывались точки контакта между поверхностями. Отметим, что в этом методе рассчитываются точки пересечения цилиндра с заданной плоскостью, когда заданы вектор направления цилиндра, радиус цилиндра и вектор нормали к плоскости.

При построении модели КА использовались лишь четыре типа элементарных фигур, для которых были написаны базовые классы, а именно грань, параллелепипед, цилиндр и сфера. Собственно язык описания представляет собой небольшое число базовых команд (лексем), содержащих информацию по построению и соединению частей. На рис. 5 показан список команд на примере класса параллелепипеда (Draw_Prld).

Окончательная модель КА "Спектр-РГ" (рис. 6, см. четвертую сторону обложки) состоит из 21 отдельного модуля, объединенных в единую связанную структуру.

Каждый модуль представляет собой набор из связанных элементарных объектов, являющихся экземплярами классов сфер, параллелепипедов и цилиндров. Модульный подход позволяет простыми средствами осуществить анимирование модели, при котором каждый из модулей может быть перемещен в любую точку пространства по заданной траектории. Для этого в языке предусмотрены лексемы-команды, аналогичные тем, что были использованы и при формировании всех исходных деталей: *MOVE*, *ROTATE*,

```
// Параллелепипед в направлении вектора Dir с центром Center
void Center(Draw_Point& Center_Point, Draw_Point& Dir);

// Привязанный к плоскости параллелепипед
void Attached(Draw_Plane* pPlane, int iOrient=0, float p1=0.5f, float p2=0.5f);

// Привязанный к двум плоскостям параллелепипед со сторонами a1, a2 Ориентация определяется
первой привязанной поверхностью
void Contact(Draw_Plane* pPlane1,  Draw_Plane* pPlane2, float p11=0.5f, float p12=0.5f, float
p21=0.5f, float p22=0.5f, bool bSmooth=true);

// Присоединиться к заданной поверхности по имеющейся нормали
void Joint_To(Draw_Plane* pPlane);

//  Соединяем верхнюю часть параллелепипеда с поверхностью, нижняя не меняется
void Joint_UpSide(Draw_Plane* pPlane, float p1, float p2, bool bSmooth=true);

// Соединяем нижнюю часть параллелепипеда с поверхностью, верхняя не меняется
void Joint_DnSide(Draw_Plane* pPlane, float p1, float p2, bool bSmooth=true);

// Точки параллелепипеда полностью совпадают с точками плоскости
void Attached_Full(Draw_Plane* pPlane); void Contact_Full(Draw_Plane* pPlane1, Draw_Plane*
pPlane2, bool bType=true);

// Добавить параллелепипед
void Add(Draw_Prld* pPrld);

// Добавить цилиндр
void Add(Draw_Tube* pTube);

// Добавить сферу
void Add(Draw_Sphere* pSphere);

// Добавить параллелепипед к верхней грани
void Add_Up(Draw_Prld* pPrld);

// Добавить цилиндр к верхней грани
void Add_Up(Draw_Tube* pTube)
....
....
```

Рис. 5. Пример методов класса параллелепипед (Draw_Prld)

JOINT и др. В результате созданная модель обладает возможностями, позволяющими пользователю путем выбора необходимой строки из основного меню открывать и закрывать солнечные элементы, поворачивать их под определенным углом к свету, а также разбирать (и собирать) модель КА на составные части.

Заключение

Разработанный набор классов и метаязык описания связей являются простым и удобным средством, позволяющим создавать "гибкие" структурированные трехмерные объекты. Базовое положение языка построения модели, состоящее в связывании отдельных частей или объектов, является эффективным подходом, предоставляющим возможность модифицировать отдельные части модели и не заботиться о целостности всего объекта. Как следствие, при работе в команде нескольких разработчиков достаточно описать интерфейсные связи между объектами, и общую модель можно конфигурировать и просматривать на любом этапе ее создания.

Применение этого подхода позволило нескольким программистам достаточно быстро создать адекватную реальному объекту модель аппарата "Спектр-РГ", включенную в прикладную информационную систему по обработке потока телеметрической информации по температурной диагностике состояния КА.

Список литературы

1. **Миллер Т.** DirectX 9 с управляемым кодом. Программирование игр и графика, М.: КомБук, 2005. 390 с.
2. **Скляров В. А.** Язык С++ и объектно-ориентированное программирование, Минск: Вышэйшая школа, 1997. 481 с.
3. **Фленов М. Е.** DirectX и C++. Искусство программирования, СПб.: БХВ-Петербург, 2006. 384 с.
4. **Олафсен Ю.** Visual C++ 6 и MFC. Энциклопедия пользователя, СПб.: ДиаСофт, 2000. 550 с.

Meta-Language for 3D-Objects Description in Applied Information Systems

N. N. Svetushkov, e-mail: svetushkov@mai.ru, Moscow Aviation Institute (National Research University), Moscow, 125993, Russian Federation

Corresponding author:

Svetushkov Nikolaj N., Associate Professor, Moscow Aviation Institute (National Research University), Moscow, 125993, Russian Federation
E-mail: svetushkov@mai.ru

Received on November 11, 2016
Accepted on December 02, 2016

The article is devoted to the new approach to create complex three-dimensional objects, based on the principle of linking component parts. There is basic (conceptual) description of developed meta-language that is for creating three-dimensional objects with simplified graphical editor without excessive requirements for memory (RAM) and computer performance. This option is rather significant when using the software in operating systems such as Android. The main provisions of the meta-language are based on ability to establish links between the individual parts or objects, so it provides the main advantage for modifying model: one should not care about the integrity of the entire facility. The most important advantage of the development system is its open code so there is a possibility for the programmer-developer to use created objects in any programs written in C++ (when linked freely available DirectX library). This approach is based on the conjunct classes that support language tokens, so there is a clear way for including their own three-dimensional objects in applications, thus increasing the commercial attractiveness of established software. Developed and implemented in C++ this approach has shown to be effective in the creation of applied information systems that require a graphical illustration. There are some examples of the use of this language and finally three-dimensional model of the spacecraft "Spectr-RG", which was created for the special information system for processing of telemetry data.

Keywords: *3D graphics, language, applied software, DirectX, information systems*

For citation:

Svetushkov N. N. Meta-Language for 3D-Objects Description in Applied Information Systems, *Programmnaya Ingeneria*, 2017, vol. 8, no. 3, pp. 136—139.

DOI: 10.17587/prin.8.136-139

References

1. **Miller T.** *DirectX 9 s upravljaemym kodom. Programmirovanie igr i grafika* (Managed DirectX 9. Games and Graphics Programming), Moscow, KomBuk, 2005, 390 p. (in Russian).
2. **Skljarov V. A.** *Jazyk C++ i obektno-orientirovannoe programmirovanie* (C++ and Object-oriented Programming), Minsk, Vyshjejshaja shkola, 1997, 481 p. (in Russian).
3. **Flenov M. E.** *DirectX i C++. Iskusstvo programmirovanija* (DirectX and C++. The Art of Programming), Saint Peterburg, BHV, 2006, 384 p. (in Russian).
4. **Olafsen E., Scibmer K., White D.** *Visual C++ 6 i MFC* (Visual C++ 6 and MFC), Saint Peterburg, DiaSoft, 2000, 718 p. (in Russian)

УДК 512.563.6 DOI: 10.17587/prin.8.140-144

И. Ю. Иванов, аспирант, e-mail: hour1scorp@gmail.com,
Воронежский государственный университет

Применение контрапозиционного правила вывода при решении продукционно-логических уравнений на булевой решетке

Алгебраический подход к интерпретации логики продукций нулевого порядка основывается на моделировании логического вывода с помощью отыскания связей между элементами математической решетки. Рассмотренная в предыдущих работах автора концепция продукционно-логических уравнений на булевой решетке развивает этот алгебраический подход, который, однако, имеет недостаток, связанный с игнорированием контрапозиционного правила вывода. В настоящей работе представлены результаты исследований, устраняющие этот недостаток.

Ключевые слова: *логика нулевого порядка, LP-структура, LP-вывод, булева решетка, правила вывода, логическое отношение, начальное множество решетки, продукционно-логическое уравнение*

Введение

LP-вывод — процесс выявления логических связей между элементами математической решетки, порожденных дополнительным бинарным отношением на этой решетке. Данное понятие основывается на концепции продукционно-логических уравнений [1] в рамках общей теории LP-структур [2]. Основу LP-структуры составляет математическая решетка. Различные типы решеток порождают соответственно различные типы LP-структур, например, LP-структуры на дистрибутивной решетке, LP-структуры на полной решетке и т. д. Выбор типа решетки зависит от прикладных задач, в которых данная теория применяется.

В работе [3] рассмотрен алгебраический подход к интерпретации продукционной логики нулевого порядка. Он основан на LP-структуре нулевого порядка — булевой решетке с дополнительно заданным на ней бинарным отношением (именуемым логическим). Такой подход получил дальнейшее развитие в работах [4, 5], где было введено понятие продукционно-логических уравнений на булевой решетке, а также предложен метод решения таких уравнений, что соответствует выводу в логике нулевого порядка. При этом упомянутый метод не использует контрапозиционное правило LP-вывода, столь же естественно отражающее правила вывода в логике нулевого порядка, как и остальные правила LP-вывода, определенные в работе [3].

В настоящей статье проведено дополнительное исследование процесса LP-вывода на булевой решетке, показывающее, каким образом можно применять контрапозиционное правило при решении продукционно-логических уравнений.

1. Логические отношения на булевых решетках

Рассмотрим конечную булеву решетку $\mathbb{F}$. Как известно [6], конечная решетка является булевой тогда и только тогда, когда она изоморфна множеству всех подмножеств (булеану) некоторого конечного множества. Таким образом, без ограничения общности можно считать, что $\mathbb{F}$ — булеан конечного множества $F = \{x_1, ..., x_n\}$, $n > 0$. Элементы $\mathbb{F}$ будем обозначать большими латинскими буквами, как это принято в теории множеств (если не оговорено иное). Рассматриваемая решетка частично упорядочена отношением $\subseteq$ нестрогого вложения множеств. Операции пересечения, объединения и дополнения на $\mathbb{F}$ есть теоретико-множественные операции $\cap, \cup, \div$ соответственно. Всякий элемент $X \in \mathbb{F}$, $X \neq F$ имеет единственное представление в виде пересечения коатомов — $(n-1)$-элементных подмножеств множества F (будем обозначать коатомы малыми латинскими буквами). Такое представление называется коразложением. Множество коатомов, составляющих коразложение X, обозначается $Coat(X)$. По определению $Coat(F) = \varnothing$. Множество всех коатомов решетки $\mathbb{F}$ обозначим $\mathcal{C}$.

Замечание 1.1. Для любого элемента $X \in \mathbb{F}$ справедливо соотношение $Coat(\overline{X}) = \overline{Coat(X)}$ [6] (дополнение в данном случае понимается в смысле универсума $\mathcal{C}$).

Бинарное отношение R (в дальнейшем будем рассматривать только бинарные отношения и говорить просто "отношение") на $\mathbb{F}$ называют логическим, если оно:

- содержит отношение $\subseteq$;

- контрапозиционно, т. е. из $(A, B) \in R$ следует $(\bar{B}, \bar{A}) \in R$;
- дистрибутивно, т. е. из (A, B_1), $(A, B_2) \in R$ следует $(A, B_1 \cap B_2) \in R$ и из (A_1, B), $(A_2, B) \in R$ следует $(A_1 \cup A_2, B) \in R$;
- транзитивно.

Логическим замыканием произвольного отношения R на $\mathbb{F}$ называют наименьшее логическое отношение, содержащее R. Два отношения R_1, R_2 на $\mathbb{F}$ называют логически эквивалентными (обозначение $R_1 \sim R_2$), если они имеют общее логическое замыкание.

Пусть R — отношение на $\mathbb{F}$. Пару (A, B), $A, B \in \mathbb{F}$ (не обязательно принадлежащую R) называют логически связанной отношением R [3] (при этом применяется обозначение $A \xrightarrow{R} B$), если выполнено одно из следующих условий:

- $$(A, B) \in R; \quad (1)$$
- $$A \subseteq B; \quad (2)$$
- $$\bar{B} \xrightarrow{R} \bar{A}; \quad (3)$$
- существуют $B_1, B_2 \in \mathbb{F}$ такие, что

$$B_1 \cap B_2 = B \text{ и } A \xrightarrow{R} B_1, \quad A \xrightarrow{R} B_2; \quad (4)$$

- существуют $A_1, A_2 \in \mathbb{F}$ такие, что

$$A_1 \cup A_2 = A \text{ и } A_1 \xrightarrow{R} B, \quad A_2 \xrightarrow{R} B; \quad (5)$$

- существует $C \in \mathbb{F}$ такой, что

$$A \xrightarrow{R} C, \quad C \xrightarrow{R} B. \quad (6)$$

Заметим, что данное определение носит рекурсивный характер. Уровнем рекурсии в паре $A \xrightarrow{R} B$ называют число применений правил (3)—(6) (возможно, к конечному множеству пар), необходимых для получения этой логической связи.

В работе [3] показано, что для произвольного отношения R на $\mathbb{F}$ его логическое замыкание существует и совпадает со множеством всех пар, логически связанных отношением R, т. е. совпадает с отношением $\xrightarrow{R}$.

Наряду с отношением $\xrightarrow{R}$ рассмотрим его подмножество $\xrightarrow{R}_0$, образованное по тем же правилам (1)—(6), за исключением контрапозиционного правила (3). При разработке понятия продукционно-логических уравнений на булевых решетках в работе [4] использовалось именно отношение $\xrightarrow{R}_0$. Рассмотрим другое, эквивалентное нерекурсивное определение отношения $\xrightarrow{R}_0$ из работы [8], которое понадобится в дальнейшем.

Пара (X, Y), $X, Y \in \mathbb{F}$ (не обязательно принадлежащая R) дистрибутивно связана отношением R, если:

- существует конечное множество пар $\{(X_i, Y_i) \mid i = 1, ..., k\}$ таких, что $X_i = Y_i$ или $(X_i, Y_i) \in R$, $i = 1, ..., k$;
- существует k-арный терм p [7] такой, что $X \subseteq p(X_1, ..., X_k)$ и $p(Y_1, ..., Y_k) \subseteq Y$.

Пара (A, B), $A, B \in \mathbb{F}$ принадлежит $\xrightarrow{R}_0$ тогда и только тогда, когда существует конечный упорядоченный набор $(B_0, B_1, ..., B_m, B_{m+1})$ элементов из $\mathbb{F}$, где $B_0 = A$, $B_{m+1} = B$ такой, что каждая пара (B_i, B_{i+1}), $i = 0, ..., m$ дистрибутивно связана отношением R.

2. Применение контрапозиционного правила вывода

Как было упомянуто ранее, понятие продукционно-логических уравнений на булевой решетке основывается на отношении $\xrightarrow{R}_0$, при построении которого не используется естественное для логики нулевого порядка контрапозиционное правило (3). В данном разделе докажем теорему, которая позволяет перейти от отношения $\xrightarrow{R}_0$ к "полному" отношению $\xrightarrow{R}$ в упомянутых уравнениях. При этом результаты работы [5], касающиеся метода получения точного решения таких уравнений, останутся актуальными.

Обозначим c — свойство отношения S на $\mathbb{F}$ "быть контрапозиционным" и $c(S)$ — контрапозиционное замыкание S (т. е. наименьшее контрапозиционное отношение на $\mathbb{F}$, содержащее S в качестве подмножества). Очевидно, что для построения контрапозиционного замыкания произвольного отношения S необходимо добавить в S недостающие пары $(\bar{B}, \bar{A})$ для каждой исходной пары $(A, B) \in S$.

Лемма 2.1. *Для произвольного отношения R на $\mathbb{F}$ справедливо равенство* $\xrightarrow{c(R)}_0 = c\left(\xrightarrow{R}_0\right)$.

Доказательство. Необходимо показать, что рассматриваемые в равенстве множества есть подмножества друг друга.

I. Рассмотрим пару $A \xrightarrow{c(R)}_0 B$. Покажем, что $(A, B) \in c\left(\xrightarrow{R}_0\right)$. Проведем доказательство с помощью индукции по оценке уровня рекурсии m в паре $A \xrightarrow{c(R)}_0 B$.

Если $m = 0$, то рассматриваемая пара получена с помощью правил (1) или (2). В случае с правилом (1) имеем $(A, B) \in c(R)$, откуда получаем две возможности. Первая — $(A, B) \in R$. В этом случае $(A, B) \in c\left(\xrightarrow{R}_0\right)$ очевидно выполняется. Вторая — $(\bar{B}, \bar{A}) \in R$. В этом случае справедливо $\bar{B} \xrightarrow{R}_0 \bar{A}$, откуда имеем $(A, B) \in c\left(\xrightarrow{R}_0\right)$. В случае с правилом (2) имеем $A \subseteq B$, откуда сразу следует $(A, B) \in c\left(\xrightarrow{R}_0\right)$.

Предположим, что все пары из $\xrightarrow{c(R)}_0$, имеющие уровень рекурсии, не превосходящий некоторого m, принадлежат $c\left(\xrightarrow{R}_0\right)$. Покажем, что аналогичное справедливо и для пары $A \xrightarrow{c(R)}_0 B$, имеющей уровень рекурсии $m + 1$. В данном случае пара была получена с использованием одного из правил (4)—(6).

Если было использовано правило (4), то существуют пары $A \xrightarrow{c(R)}_0 B_1$, $A \xrightarrow{c(R)}_0 B_2$ такие, что $B_1 \cap B_2 = B$. Так как последние две пары имеют уровень рекурсии, не превосходящий m, то $(A, B_1) \in c\left(\xrightarrow{R}_0\right)$, $(A, B_2) \in c\left(\xrightarrow{R}_0\right)$ по предположению. Тогда $(A, B_1 \cap B_2) \in c\left(\xrightarrow{R}_0\right)$, что в точности означает $(A, B) \in c\left(\xrightarrow{R}_0\right)$.

Если было использовано правило (5), то существуют пары $A_1 \xrightarrow{c(R)}_0 B$, $A_2 \xrightarrow{c(R)}_0 B$ такие, что $A_1 \cup A_2 = A$. Так как последние две пары имеют уровень рекурсии, не превосходящий m, то $(A_1, B) \in c\left(\xrightarrow{R}_0\right)$, $(A_2, B) \in c\left(\xrightarrow{R}_0\right)$ по предположению. Тогда $(A_1 \cup A_2, B) \in c\left(\xrightarrow{R}_0\right)$, что вновь в точности означает $(A, B) \in c\left(\xrightarrow{R}_0\right)$.

Наконец, если было использовано правило (6), то существуют пары $A \xrightarrow{c(R)}_0 D$, $D \xrightarrow{c(R)}_0 B$. Поскольку данные пары имеют уровень рекурсии, не превосходящий m, то $(A, D) \in c\left(\xrightarrow{R}_0\right)$, $(D, B) \in c\left(\xrightarrow{R}_0\right)$, откуда получаем $(A, B) \in c\left(\xrightarrow{R}_0\right)$.

II. Пусть $(A, B) \in c\left(\xrightarrow{R}_0\right)$. Необходимо показать, что $A \xrightarrow{c(R)}_0 B$.

Возможны два случая: $A \xrightarrow{R}_0 B$ и $\overline{B} \xrightarrow{R}_0 \overline{A}$. Так как $R \subseteq c(R)$, то $\xrightarrow{R}_0 \subseteq \xrightarrow{c(R)}_0$. Первый случай тривиален, поскольку он сразу влечет $A \xrightarrow{c(R)}_0 B$. Далее рассмотрим второй случай.

Приведем нерекурсивное определение логической связи $\overline{B} \xrightarrow{R}_0 \overline{A}$, рассмотренное в предыдущем разделе. В данном случае оно означает, что существует упорядоченный набор $(E_0, E_1, ..., E_m, E_{m+1})$ элементов из $\mathbb{F}$ такой, что $E_0 = \overline{B}$ и $E_{m+1} = \overline{A}$, причем все пары (E_i, E_{i+1}), $i = 0, ..., m$ дистрибутивно связаны отношением R. В свою очередь это означает, что существуют множества пар $\{(P_{ij}, Q_{(i+1)j} \mid j = 1, ..., m_i\}$ такие, что $P_{ij} = Q_{(i+1)j}$ или $(P_{ij}, Q_{(i+1)j}) \in R$, $i = 1, ..., m$, и также существуют m_i-арные термы p_i такие, что $E_i \subseteq p_i\left(P_{i1}, ..., P_{im_i}\right)$ и $p_i\left(Q_{(i+1)1}, ..., Q_{(i+1)m_i}\right) \subseteq E_{i+1}$, $i = 1, ..., m$.

Отсюда следует, что существуют множества пар $\left\{\left(\overline{Q_{(i+1)j}}, \overline{P_{ij}}\right) \mid j = 1, ..., m_i\right\}$ такие, что $\overline{Q_{(i+1)j}} = \overline{P_{ij}}$ или $\left(\overline{Q_{(i+1)j}}, \overline{P_{ij}}\right) \in c(R)$, $i = 1, ..., m$, и также существуют m_i-арные термы q_i такие, что $\overline{E_{i+1}} \subseteq q_i\left(\overline{Q_{(i+1)1}}, ..., \overline{Q_{(i+1)m_i}}\right)$ и $q_i\left(\overline{P_{i1}}, ..., \overline{P_{im_i}}\right) \subseteq \overline{E_i}$, $i = 1, ..., m$. Здесь q_i — такой терм, который принимает значение $\overline{p_i\left(R_1, ..., R_{m_i}\right)}$ на наборе аргументов $\left(\overline{R_1}, ..., \overline{R_{m_i}}\right)$, $i = 1, ..., m$. В свою очередь это означает, что существует упорядоченный набор $\left(\overline{E_{m+1}}, \overline{E_m}, ..., \overline{E_1}, \overline{E_0}\right)$ такой, что каждая пара $\left(\overline{E_{i+1}}, \overline{E_i}\right)$, $i = 1, ..., m$ дистрибутивно связана отношением $c(R)$. Отсюда получаем $\overline{E_{m+1}} \xrightarrow{c(R)}_0 \overline{E_0}$, что в точности означает $A \xrightarrow{c(R)}_0 B$. ■

В работе [2] была доказана следующая полезная лемма.

Лемма 2.2. *Пусть R — отношение на $\mathbb{F}$. Тогда при выводе любой логической связи $A \xrightarrow{R} B$ все применения контрапозиционного правила (3) могут быть исключены либо перенесены в начальную стадию этого процесса.*

Наконец, сформулируем и докажем основную теорему настоящей работы — о применении контрапозионного правила (3).

Теорема 2.1. *Пусть R — отношение на $\mathbb{F}$. Тогда при выводе любой логической связи $A \xrightarrow{R} B$ все применения контрапозиционного правила (3) могут быть исключены либо перенесены в начальную, равно как и в заключительную, стадию этого процесса.*

Доказательство. Истинность первой части теоремы, утверждающей, что использование правила (3). может быть исключено или перенесено в начальную стадию процесса вывода, следует непосредственно из леммы 2.2. Докажем оставшуюся часть теоремы.

Заметим, что утверждение леммы 2.2 может быть записано в виде равенства $\xrightarrow{R} = \xrightarrow{c(R)}_0$. Вместе с тем в лемме 2.1 утверждается, что $\xrightarrow{c(R)}_0 = c\left(\xrightarrow{R}_0\right)$. Используя два последних равенства, получаем новое равенство $\xrightarrow{R} = c\left(\xrightarrow{R}_0\right)$, которое и означает, что применение правила (3) может быть также перенесено в конечную стадию процесса вывода. ■

3. Применение контрапозиционного правила при решении уравнений

Рассмотрим, каким образом можно перейти от отношения $\xrightarrow{R}_0$ к отношению $\xrightarrow{R}$ при определении понятия продукционно-логического уравнения на булевой решетке и разработке методов решения таких уравнений.

Замечание 3.1. Пусть R_{cp} — контрапозиционное отношение на $\mathbb{F}$, т. е. $c(R_{cp}) = R_{cp}$. Тогда, используя теорему 2.1, можно получить равенство $\xrightarrow{R_{cp}}_0 = \xrightarrow{c(R_{cp})}_0 = \xrightarrow{R_{cp}}$. То есть, если исходное отношение R на решетке $\mathbb{F}$ уже является контрапозиционным, то получение логических связей по правилам построения отношения $\xrightarrow{R}_0$ эквивалентно получению логических связей по правилам построения отношения $\xrightarrow{R}$, поскольку приводит к одинаковому результату. Таким образом, для применения контрапозиционного правила (3) в рамках существующей теории продукционно-логических уравнений на булевой решетке достаточно исходное отношение R заменить его контрапозиционным

замыканием $c(R)$ (что достигается простым добавлением недостающих "контрпар").

Одно из базовых понятий в теории продукционно-логических уравнений, с помощью которого вводится понятие решения уравнения, есть понятие начального множества. В случае с уравнениями на булевой решетке оно определяется следующим образом [8].

Коатом $x \in \mathbb{F}$ называют начальным при отношении R, если для любой пары $A \xrightarrow{R}_0 B$ из того, что $B = B_1 \cap x$, следует $A = A_1 \cap x$ при некоторых A_1, $B_1 \in \mathbb{F}$. Начальным множеством решетки $\mathbb{F}$ при отношении R называют нижнюю полурешетку $\mathbb{F}_0(R)$, образованную всевозможными пересечениями начальных при отношении R коатомов решетки $\mathbb{F}$.

Выясним, каким образом замена отношения $\xrightarrow{R}_0$ отношением $\xrightarrow{R}$ повлияет на построение начального при отношении R множества решетки. Для этого рассмотрим понятие канонической формы отношения на $\mathbb{F}$ [8].

Канонической формой отношения R на $\mathbb{F}$ называется отношение R_{cn}, эквивалентное R, правые части пар которого есть коатомы, не содержащее пар отношения $\subseteq$. Для построения канонической формы отношения R необходимо заменить каждую его пару (A, B) на множество пар$\{(A, b) \mid b \in Coat(B)\}$, а затем из полученного отношения исключить пары отношения $\subseteq$. Будем обозначать $\mathcal{C}(R_{cn})$ — множество коатомов из правых частей канонического отношения R_{cn}.

Замечание 3.2. В работе [8] показано, что множество начальных при отношении R коатомов решетки $\mathbb{F}$ совпадает со множеством $\mathcal{C} \setminus \mathcal{C}(R_{cn})$.

Лемма 3.1. *Для канонических форм R_{cn} и $c(R)_{cn}$ отношений R и $c(R)$ на решетке $\mathbb{F}$ соответственно справедливо равенство $\mathcal{C}(R_{cn}) = \mathcal{C}(c(R)_{cn})$.*

Доказательство. Необходимо показать два вложения: $\mathcal{C}(R_{cn}) \subseteq \mathcal{C}(c(R)_{cn})$ и $\mathcal{C}(c(R)_{cn}) \subseteq \mathcal{C}(R_{cn})$.

Докажем первое вложение. Поскольку $R \subseteq c(R)$, то $R_{cn} \subseteq c(R)_{cn}$. Следовательно, для любой пары $(X, y) \in R_{cn}$ справедливо $(X, y) \in c(R)_{cn}$. Это означает, что $\mathcal{C}(R_{cn}) \subseteq \mathcal{C}(c(R)_{cn})$.

Докажем обратное вложение. Пусть $(X, y) \in c(R)_{cn}$. Тогда существует пара $(X, Y) \in c(R)$ такая, что $y \in Coat(Y)$. Возможны два случая: $(X, Y) \in R$ и $(\bar{Y}, \bar{X}) \in R$. В первом случае, когда $(X, Y) \in R$, сразу получаем $(X, y) \in R_{cn}$. Рассмотрим второй случай: $(\bar{Y}, \bar{X}) \in R$. По замечанию (1) справедливы равенства $Coat(\bar{X}) = \overline{Coat(X)}$, $Coat(\bar{Y}) = \overline{Coat(Y)}$. Так как отношение $c(R)_{cn}$ каноническое, то $X \not\subseteq y$, следовательно, $y \notin Coat(X)$ [8]. Таким образом, справедливы соотношения: $y \in Coat(\bar{X})$, $y \notin Coat(\bar{Y})$. Отсюда получаем $(\bar{Y}, y) \in R_{cn}$. В обоих случаях наличие пары с коатомом y в правой части в отношении $c(R)_{cn}$ влечет наличие некоторой пары с тем же коатомом y в правой части в отношении R_{cn}, т. е. $\mathcal{C}(c(R)_{cn}) \subseteq \mathcal{C}(R_{cn})$.

Таким образом, окончательно имеем $\mathcal{C}(R_{cn}) = \mathcal{C}(c(R)_{cn})$. ■

Следствие 3.1. Начальные множества решетки $\mathbb{F}$ при отношениях R и $c(R)$ совпадают.

Доказательство. По замечанию 3.2 множество начальных при отношении R коатомов решетки $\mathbb{F}$ есть множество $\mathcal{C} \setminus \mathcal{C}(R_{cn})$. С учетом замечания 3.1, по замечанию 3.2 также получаем, что множество начальных при отношении $c(R)$ коатомов решетки $\mathbb{F}$ есть множество $\mathcal{C} \setminus \mathcal{C}(c(R)_{cn})$. Тогда, с учетом леммы 3.1, имеем $\mathcal{C} \setminus \mathcal{C}(R_{cn}) = \mathcal{C} \setminus \mathcal{C}(c(R)_{cn})$. Отсюда, из определения начального множества, получаем $\mathbb{F}_0(R) = \mathbb{F}_0(R_{cn})$. ■

Следствие 3.1 показывает, что использование контрапозиционного правила (3) не меняет начальное множество $\mathbb{F}_0(R)$.

Заключение

Использование контрапозиционного правила является неотъемлемой частью процесса продукционно-логического вывода. В то же время при разработке концепции логических уравнений на булевой решетке в работах [4, 5, 8] применение этого правила не рассматривалось. Настоящая работа устраняет отмеченный недостаток. Доказанная теорема позволяет использовать все ранее полученные результаты, касающиеся упомянутых уравнений, заменив лишь исходное отношение на решетке его контрапозиционным замыканием. Показано также, что применение контрапозиционного правила не изменяет начальное множество решетки при заданном отношении.

Практическая значимость проведенного исследования связана с реализацией алгебраического подхода к интерпретации продукционной логики нулевого порядка [3], а также с применением данного подхода к построению интеллектуальных продукционно-логических систем [9], правила которых оперируют логическими связками пропозиционального языка нулевого порядка. Использование полного набора правил вывода позволит снизить число обращений к внешним источникам информации, например, к экспертам, путем рационального использования всех знаний, формализованных в исходном множестве продукционных правил.

Данная работа поддержана грантом РФФИ № 15-07-05341.

Список литературы

1. **Махортов С. Д.** Логические уравнения на решетках // Вестник ВГУ. Серия Физика, математика. 2004. № 2. С. 170—178.

2. **Махортов С. Д.** Математические основы искусственного интеллекта: Теория LP-структур для построения и исследования моделей знаний продукционного типа / Под ред. В. А. Васенина. М.: МЦНМО, 2009. 304 с.

3. **Махортов С. Д.** Об алгебраической интерпретации продукционной логики нулевого порядка // Вестник ВГУ. Серия Системный анализ и информационные технологии. 2007. № 1. С. 56—63.

4. **Иванов И. Ю.** Продукционно-логические уравнения на булевых решетках // Вестник ВГУ. Серия Физика. Математика. 2013. № 1. С. 170—177.

5. **Иванов И. Ю.** Расширенная модель LP-вывода на булевой решетке // Программная инженерия. 2016. Т. 7, № 6. С. 252—257.

6. **Биркгоф Г.** Теория решеток: пер. с англ. М.: Наука, гл. ред. физ.-мат. лит., 1984. 568 с.

7. **Гретцер Г.** Общая теория решеток: пер. с англ. / Под ред. Д. М. Смирнова. М.: Мир, 1981. 456 с.

8. **Иванов И. Ю.** Приближенное решение продукционно-логического уравнения на булевой решетке // Нейрокомпьютеры: разработка, применение. 2014. № 10. С. 53—63.

9. **Чечкин А. В.** Математическая информатика. М.: Физматлит, 1991. 416 с.

Contrapositive Inference Rule Application in Solving of Production-Logical Equations on Boolean Lattice

I. Yu. Ivanov, hour1scorp@gmail.com, Voronezh State University, Voronezh, 394018, Russian Federation

Corresponding author:

Ivanov Il'ya Yu., Postgraduate Student, Voronezh State University, Voronezh, 394018, Russian Federation
E-mail: hour1scorp@gmail.com

Received on December 03, 2016
Accepted on December 20, 2016

An algebraic approach to zero-order production logic interpretation is based on modelling inference process by finding connections between mathematical lattice's elements (LP-inference). An algebraic structure arising in this approach is an LP-structure — boolean lattice with additional binary relation defined over it. An additional relation is built using recursive rules that reflect natural rules of logical inference.

Author's previous works advance this approach by introducing the concept of production-logical equations on boolean lattice and proposing the method for solving such equations which corresponds to inference in zero-order logic. This method ignores the usage of contrapositive inference rule that is natural for zero-order logic as the other inference rules defined in LP-structures theory.

This paper provides additional research on inference process bringing the ability to use the contrapositive rule to the concept of production-logical equations on boolean lattice. The theorem proved allows applying the previously obtained results regarding production-logical equations on boolean lattice without amendments by simply replacing an initial relation over lattice with its contrapositive closure. It is also shown that such a replacement does not affect an initial set of lattice.

The practical significance of the provided study is connected with applying production-logical equations for building production systems that use logical connections of zero-order propositional language in its rules. Adding contrapositive rule to the process of inference will allow one to reduce the number of queries to the external source of information by exhaustive usage of knowledge from an initial set of rules.

Keywords: *zero-order logic, LP-structure, LP-inference, boolean lattice, inference rules, logical relation, initial set of lattice, production-logical equation*

Achnowledgements: *This work has been supported by grant 15-07-05341 from the Russian Foundation for Basic Research.*

For citation:

Ivanov I. Yu. Contrapositive Inference Rule Application in Solving of Production-Logical Equations on Boolean Lattice, *Programmnaya Ingeneria,* 2017, vol. 8, no. 3, pp. 140—144.

DOI: 10.17587/prin.8.140-144

References

1. **Mahortov S. D.** Logicheskie uravnenija na reshetkah (Logical Relations on Lattices), *Vestnik VGU. Serija Fizika, matematika*, 2004, no. 2, pp. 170—178 (in Russian).

2. **Mahortov S. D.** *Matematicheskie osnovy iskusstvennogo intellekta: teorija LP-struktur dlja postroenija i issledovanija modelej znanij produkcionnogo tipa* (Mathematical Foundations of Artificial Intelligence: Theory of LP-structures for the Construction and Studying of Knowledge Models of Production Type), Moscow, MCNMO, 2009, 304 p. (in Russian).

3. **Mahortov S. D.** Ob algebraicheskoj interpretacii produkcionnoj logiki nulevogo porjadka (About an Algebraic Interpretation of Zero-order Production Logic), *Vestnik VGU. Serija Sistemnyj analiz i informacionnye tehnologii,* 2007, no. 1, pp. 56—63 (in Russian).

4. **Ivanov I. Yu.** Produkcionno-logicheskie uravnenija na bulevyh reshetkah (Production-logical Equations on Boolean Lattices), *Vestnik VGU. Serija Fizika. Matematika,* 2013, no. 1, pp. 170—177 (in Russian).

5. **Ivanov I. Yu.** Rasshirennaja model' LP-vyvoda na bulevoj reshjotke (Extended Model of LP-inference on Boolean Lattice), *Programmnaya Ingeneria*, 2016, vol. 7, no. 6, pp. 252—257 (in Russian).

6. **Birkgof G.** *Teorija reshjotok* (Lattice Theory), Moscow, Nauka, glavnaja redakcija fiziko-matematicheskoj literatury, 1984, 568 p. (in Russian).

7. **Gratzer G.** *Obshhaja teorija reshjotok* (General Lattice Theory), Moscow, Mir, 1981, 456 p. (in Russian).

8. **Ivanov I. Y.** Priblizhennoe reshenie produkcionno-logicheskogo uravnenija na bulevoj reshetke (Particular Solution of Production-logical Equation on Boolean Lattice), *Nejrokomp'jutery: razrabotka, primenenie*, 2014, no. 10, pp. 53—63 (in Russian).

9. **Chechkin A. V.** *Matematicheskaja informatika* (Mathematical Informatics), Moscow, Nauka, glavnaja redakcija fiziko-matematicheskoj literatury, 1991, 416 p. (in Russian).

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *Е. В. Комиссарова*

Сдано в набор 29.12.2016 г. Подписано в печать 17.02.2017 г. Формат 60×88 1/8. Заказ PI317
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**