УДК 004.89
DOI: 10.18101/2304-5728-2018-4-22-36

МЕТОД ОПИСАНИЯ СИСТЕМ СО СЛОЖНЫМ ПОВЕДЕНИЕМ НА ПРИНЦИПАХ ОБОБЩЕННЫХ АВТОМАТОВ

© **Федорченко Людмила Николаевна**
кандидат технических наук, доцент,
старший научный сотрудник,
Санкт-Петербургский институт информатики и автоматизации
Российской академии наук
Россия, 199178, г. Санкт-Петербург, 14 Линия В.О., 39
E-mail:LNF@iias.spb.su

© **Афанасьева Ирина Викторовна**
ведущий инженер,
Специальная астрофизическая обсерватория Российской академии наук
Россия, 369167, Карачаево-Черкесская Республика, п. Нижний Архыз
E-mail:riv@sao.ru

Автоматные модели наиболее удобны для проектирования систем со сложным поведением, но имеют ряд ограничений, таких как отсутствие поддержки асинхронности и параллельности. Для описания таких моделей часто используют специальные языковые средства, как графические, так и текстовые. В предлагаемой статье представлен метод описания систем со сложным поведением с использованием языка программирования высокого уровня CIAO (Cooperative Interaction of Automata Objects), из программы которого генерируется система автоматов, симулирующих функционирование исходной реагирующей системы.

***Ключевые слова**:* асинхронные параллельные реагирующие системы; граф переходов состояний; синтаксическая граф-схема; грамматика в регулярной форме.

Введение

Большой класс технологических инструментов для производства программ обработки данных использует принцип синтаксического управления. В типичных приложениях, таких как трансляция языков программирования, этот принцип давно и успешно используется. Управление процессом трансляции определяется синтаксической структурой предложений входного языка.

Особенность таких программ состоит в том, что структура входных данных (входного языка) и система управления обработкой данных такой программы определяется сходным образом: одной и той же формальной грамматикой, или системой конечных автоматов, или же управляющей таблицей. Главная часть логики такой программы сосредоточена в схеме взаимодействующих автоматов, а сама программа обращается к ней на каждом шаге ее работы. Управляющая таблица, в частности, определяет порядок вызова семантических процедур, осуществляющих обработку данных в процессе анализа входного текста.

Цель исследования, которому частично посвящена статья, разработать методику автоматизированного формирования событийно-управляемой программной системы с помощью языка CIAO (Cooperative Interaction of Automata Objects) [1–3] для проектирования асинхронных параллельных реагирующих систем.

Краткая историческая справка

Методы разработки сложных программных систем определяются используемой моделью поведения. В известной классификации поведенческих моделей программ Д. Харела выделяются *реагирующие* и *трансформационные системы* [4]. Большинство систем управления техническими средствами являются реагирующими.

Разработанное специальное программное обеспечение сбора и обработки данных от астрономических фотоприемных устройств (СПО ФПУ) взаимодействует с внешней средой посредством обмена сообщениями [2]. Оно может выполнять одновременно несколько независимых операций, поэтому рассматривается как параллельное приложение. Эффективность достигается за счет использования компьютера с несколькими или многоядерными процессорами. Сложность использования многоядерных и многопроцессорных структур заключается в необходимости применения специальных средств программирования, так как первоначально программы не являются параллельным представлением алгоритмов решения задач.

С целью преодоления этих трудностей создан графический язык CIAO [3], который предназначен прежде всего:

- для наглядного описания и визуализации систем со сложным поведением;
- реализации взаимодействия параллельных программных объектов;
- построения моделей систем со сложным поведением, исследования их и верификации;
- быстрого создания прототипов параллельных реагирующих систем.

Язык CIAO показал хорошие результаты применительно к СПО ФПУ, в связи с чем возникла идея расширить область применения этого языка. Далее, будем рассматривать посылку и прием сигналов как абстрактные операции, которые выполняются системой взаимодействующих параллельных автоматов. Отсюда и интерес к языку CIAO, на котором можно писать программы для сети взаимодействующих автоматных объектов с возможным расширением языка за счет добавления новых конструкций, специфических для конкретных предметных областей.

Для применения в широком классе предметных областей с возможностью настройки языка на предметную область (кастомизация) предпочтительнее текстовая форма языка, поскольку она открывает возможность построения синтаксически управляемых программ обработки входных

структурированных данных (процессоров), о которых шла речь выше. В качестве перспективной формальной базы для построения процессоров разработана методика, основанная на использовании грамматик с обобщенными регулярными выражениями и атрибутами в виде специальных символов, семантик, для представления контекстных ограничений в синтаксической граф-схеме [6; 7].

Формальными моделями методики построения языковых процессоров, моделирующих кооперативное взаимодействие автоматных объектов, являются регулярные выражения — как средство описания языка, конечный автомат — в качестве адекватного средства его обработки и его обобщения в виде диаграмм переходов состояний.

Теоретическим обоснованием этих моделей являются теорема С. Клини и ее следствия о том, что класс регулярных множеств является минимальным классом, содержащим все конечные множества, замкнутым относительно операций объединения, конкатенации и замыкания [8].

Регулярные множества распознаются конечными автоматами и представляются регулярными выражениями.

Использование этой модели ограничено ее применением лишь к регулярным языкам, поэтому модель обобщена до класса контекстно-свободных языков в регулярной форме (КСР-языков) с использованием специальных автоматов [5]. КСР-язык порождается КСР-грамматикой — обобщением контекстно-свободной грамматики (КС-грамматики).

Основные определения даны в работах [6–8]. Напомним их неформально.

Как известно, формальным языком является набор цепочек языковых токенов (лексем или терминалов) в соответствии с грамматикой языка [8]. Грамматика языка состоит из контекстно-свободной части в виде КС-правил и контекстно-зависимой части (ограничений). КС-правила представлены в форме с обобщенными регулярными выражениями. Обобщение конечно-автоматной модели обработки языков сводится к следующему:

- вводится итерация, обозначаемая знаком «диез» (#) или обобщенная итерация, которая не расширяет множество регулярных слов и может быть определена через традиционную (одноместную) операцию Клини (*) как (P#Q) = P, (Q, P)*. Она удобна при работе со стеком и частично решает задачу минимизации регулярного выражения по числу вхождений символов из объединенного алфавита всех символов грамматики;
- язык описывается с помощью КСР-грамматики — обобщения КС-грамматики. Класс языков не расширяет, но снимает лишнюю структурированность грамматического описания языка, в частности, простые конструкции, например, последовательности, списки с разделителями описываются без рекурсивных правил, только с применением итерации;

• строится синтаксическая граф-схема (СГС) — графический аналог КСР-грамматики, стартовый объект для синтеза распознавателя (анализатора) языка.

Грамматическое описание языка CIAO

Правила грамматики записываются следующим образом:

Нетерминал : Регулярное_выражение,

где Нетерминал — одно из вышеперечисленных обозначений для нетерминалов, а Регулярное_выражение задается следующим синтаксисом в формализме Наура — Бэкуса:

```
Регулярное_выражение ::= {
      Пусто | Лексема | Нетерминал | Семантика |          //Базовые элементы (1)
      Регулярное_выражение1  Регулярное_выражение2 |     //Конкатенация (2)
      Регулярное_выражение1 ';' Регулярное_выражение2 |  //Альтернативный выбор (3)
      Регулярное_выражение1 '#' Регулярное_выражение2 |  //Итерация (4)
      '[' Регулярное_выражение ']'                       //Необязательный элемент (5)
      '(' Регулярное_выражение ')' }.                    //Выражение в скобках (6)
```

В фигурных скобках через вертикальную черту перечислены альтернативы. В строке (1) перечислены базовые элементы, составляющие регулярное выражение: Пусто — пустое выражение (отсутствие чего-либо), Лексема, Нетерминал и Семантика. Строка (2) представляет операцию конкатенации, не имеющую специального знака для своего обозначения; строка (3) — это операция альтернативного выбора, знаком которой является точка с запятой; строка (4) задает операцию итерации, знаком которой является диез, строка (5) задает необязательную конструкцию, то есть [P] = (P ;), а строка (6) позволяет заключать регулярное выражение в круглые скобки, чтобы рассматривать его как один операнд в операциях конкатенации, альтернативного выбора и итерации. Для обозначения комментария используется комбинация символов //.

Ниже описан конкретный синтаксис языка CIAO с минимумом разделителей. Ради краткости считается, что тип идентификаторов определяется на лексическом уровне. Интенсивно используется итерация, рекурсивные правила в явно выписанной части грамматики не используются.

Нетерминалы

В текущей версии грамматики языка CIAO выделяются следующие 15 нетерминалов для обозначения отдельных языковых структур: P (описание одного автоматного объекта на языке CIAO), N (номер и имя автоматного объекта), V (раздел переменных), E (раздел событий — входных команд), A (раздел действий — выходных команд), R (раздел выходных запросов), Q (раздел входных запросов), U (раздел устойчивых состояний), W (раздел неустойчивых состояний), D (раздел состояний выбора),

G (раздел групповых состояний), L (раздел внешних связей), B (булевское выражение), S (общий нетерминал для всех видов состояний), Action (список действий и/или выражений на переходе). Начальным нетерминалом, из которого порождается текст любой синтаксически правильной программы, является нетерминал P. Полная программа на CIAO является последовательностью разделов в фиксированном порядке.

Терминалы

Лексический анализатор преобразует цепочку символов из входного файла в последовательность лексем: несколько лексем общего вида: '<char>' (литера), '<number>' (целое), '<operation>' (знак операции), '<string>' (строка), '<tag>' (идентификатор) и ряд «частных» лексем — ключевых слов, перечень которых определяется грамматикой входного языка CIAO. Лексемы выступают в роли терминалов и заключены в кавычки. В языке CIAO можно выделить следующие группы лексем:

1) лексемы-названия разделов:

'VAR', 'EVENT', 'ACTION', 'REQUEST', 'QUERY', 'STATE', 'WAGGLY', 'DECISION', 'GROUP', 'LINK';

2) лексемы общего вида (в их обозначениях присутствуют угловые скобки < и >):

- '<p_num>' — уникальный номер автоматного объекта в пространстве имен, записывается так: #целое;
- '<p_nm>' — имя автоматного объекта, один или несколько произвольных идентификаторов;
- '<v_nm>' — имя локальной переменной, произвольный идентификатор;
- '<type>' — идентификатор встроенного типа, то есть один из идентификаторов 'string', 'Real', 'Integer', 'Boolean', или идентификатор внешней структуры (класса);
- '<expr>' — логическое или арифметическое выражение;
- '<e_nm>' — имя события — входной команды (часть предоставляемого интерфейса), произвольный идентификатор;
- '<a_nm>' — имя действия — выходной команды (часть требуемого интерфейса), произвольный идентификатор;
- '<r_nm>' — имя выходного запроса (часть требуемого интерфейса), произвольный идентификатор;
- '<q_nm>' — имя входного запроса (часть предоставляемого интерфейса), произвольный идентификатор;
- '<u_nm>' — имя устойчивого состояния, произвольный идентификатор;
- '<w_nm>' — имя неустойчивого состояния, произвольный идентификатор;

- '<d_nm>' — имя состояния выбора, идентификатор, записывается так: dcsnцелое. Под целым числом подразумевается номер, указанный в состоянии;
- '<g_nm>' — имя группового состояния, произвольный идентификатор, записывается так: grpцелое. Под целым числом подразумевается номер, указанный в состоянии;
- '<f_num>' — номер вызывающего автоматного объекта, которому данный автомат предоставляет указанный интерфейс, записывается так: #целое;
- '<t_num>' — номер вызываемого автоматного объекта, чей указанный интерфейс требуется, записывается так: #целое;
- '<val>' — самоопределенное значение: либо «строка», либо истинностное значение 'T', 'F', либо натуральное число, либо число с плавающей точкой;

3) лексемы-метасимволы языка, изображаемые одним или двумя специальными символами:

- ':' — после двоеточия указывается тип определяемой переменной или возвращаемого значения, для входного запроса указывается имя локальной переменной;
- ',' — соединяет последовательно;
- '|' — разделяет альтернативы;
- '[' ']' — выделяет сторожевое условие на переходе;
- '(' ')' — группирует;
- '=' — оператор присвоения значения;
- '->' — признак начала или окончания перехода;
- '/' – признак начала действий на переходе.

4) лексемы — зарезервированные слова: 'Real' — тип число с плавающей точкой, 'Integer' — целочисленный тип, 'Boolean' — булевский тип, 'after' — выход из состояния по прошествии указанного интервала времени, 's' — размерность в секундах, 'ms' — размерность в миллисекундах, 'entry' — начальное состояние, 'exit' — заключительное состояние, 'T' — значение «истина» для булевского типа, 'F'– значение «ложь» для булевского типа.

Для учета неформализованных контекстных зависимостей в правила грамматики в виде регулярных выражений введены следующие 39 семантик-процедур, которые исполняются, если в процессе распознавания входного текста очередная распознанная лексема является той, которая в грамматическом правиле следует за данной семантикой; причем эта семантика имеет доступ ко всем параметрам данной лексемы; название каждой семантики начинается со знака "$": `$after`, `$action1`, `$action2`, `$action3` `$action4`, `$attr1`, `$attr2`, `$attr3`, `$close`, `$comma1`, `$comma2`, `$cond1`, `$cond2`, `$d_nm1`, `$d_nm2`, `$else`, `$event`, `$g_nm`, `$link1`, `$link2`, `$open`, `$query`, `$r_nm`, `$request`, `$sign1`, `$sign2`, `$sign3`, `$sign4`, `$start`, `$state1`,

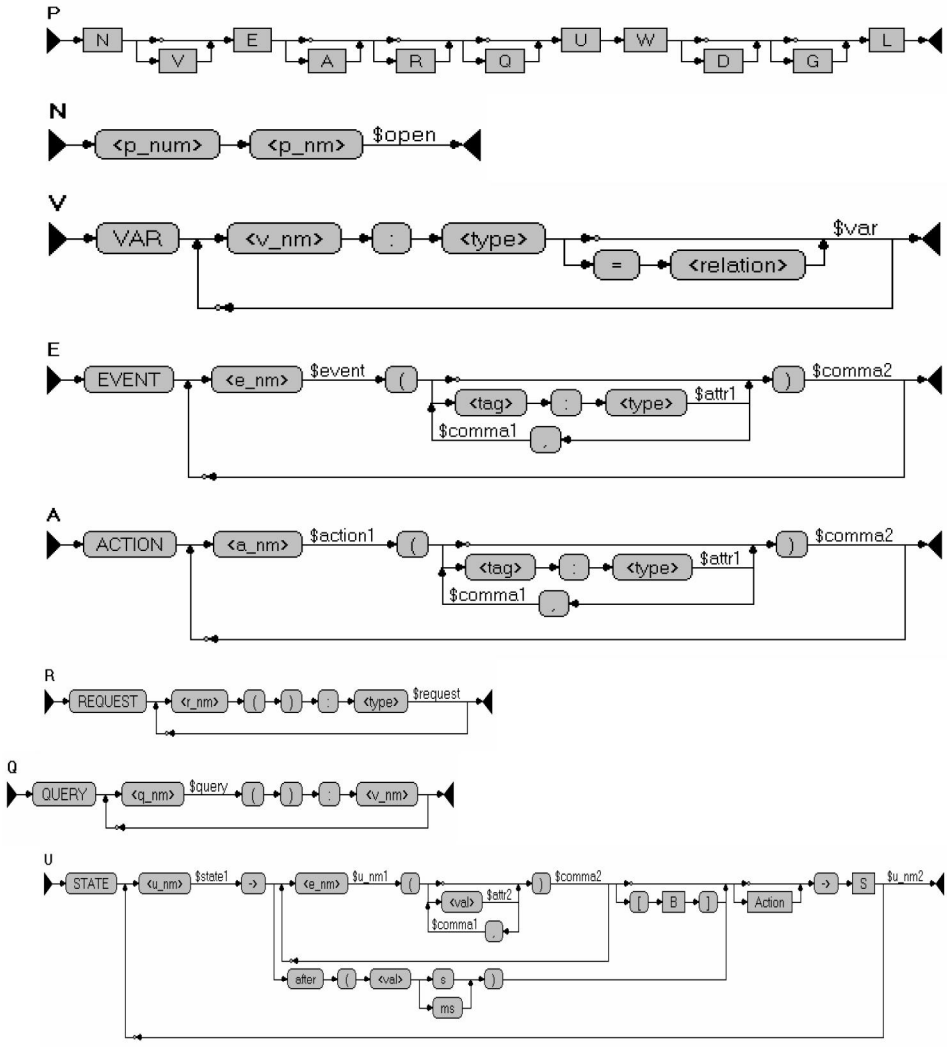
```
$state2, $state3, $state4, $state5, $state6, $u_nm1, $u_nm2,
$var, $w_nm.
```

Правила грамматики языка CIAO:

```
  P : N [V] E [A] [R] [Q] U W [D] [G] L .
  N : '<p_num>' '<p_nm>' $open .
  V : 'VAR' ( '<v_nm>' ':' '<type>' [ '=' '<expr>' ] $var
)# .
  E : 'EVENT' ( '<e_nm>' $event '(' [ ( '<tag>' ':'
'<type>'
      $attr1 )#( ',' $comma1 ) ] ')' $comma2 ) # .
  A : 'ACTION' ( '<a_nm>' $action1 '(' [ ( '<tag>' ':'
'<type>'
      $attr1 )#( ',' $comma1 ) ] ')' $comma2 ) # .
  R : 'REQUEST' ( '<r_nm>' '(' ')' ':' '<type>' $request
) # .
  Q : 'QUERY' ( '<q_nm>' $query '(' ')' ':' '<v_nm>' ) #
.
  U : 'STATE' ( ( '<u_nm>' $state1 ( '->' ( ( ( (
'<e_nm>' $u_nm1
      '(' [ ( '<val>' $attr2 )#( ',' $comma1 ) ] ')'
$comma2 )# )
      [ '[' B ']' ] ; ( 'after' '(' '<val>' ( 's' ; 'ms'
) ')'
      $after ) ) [ Action ] $action4 '->' S ) ) )# )
$u_nm2 .
  W : 'WAGGLY' ( 'entry' '->' [ '<e_nm>' $w_nm '(' [ (
'<val>'
      $attr2 )#( ',' $comma1 ) ] ')' $comma2 ] [ Action ]
$action4
      '->' '<u_nm>' $start ) #( '<w_nm>' $state2 '->'
      [ 'after' '(' '<val>' ('s' ; 'ms') ')' $after ] [
Action ]
      $action4 '->' S $state4 ) .
  D : 'DECISION' ( '<d_nm>' $d_nm1 '->' '(' ( '[' B ']' [
Action ]
      $action4 '->' S $state5 )#( '|' $else ) ( '|' $else
      [ '[' B ']' ] [ Action ] $action4 '->' S $state5 )
')' )# .
  G : 'GROUP' ( '<g_nm>' $g_nm '->' [ Action ] $action4
      '->' S $state6 )#( $d_nm2 ) .
  L : 'LINK' ( ( '<f_num>' '<p_num>' ( '<e_nm>' ;
'<q_nm>' )
      $link1 ) ; #( '<p_num>' '<t_num>' ( '<a_nm>' ;
'<r_nm>' )
      $link2 ) )#( $close ).
  B : $cond1 ( ( [ '!' $sign1 ] ( ( '<v_nm>' ( '<' ; '<='
; '>' ;
      '>=' ; '==' ; '!=' ) $sign2 '<expr>' $cond2 ) ; (
'<r_nm>'
```

```
        '(' ')' $r_nm ) ) )#( '&&' $sign3 [ '!' $sign1 ] )
)#( '||'
        $sign4 [ '!' $sign1 ] ) .
   Action : '/' ( ( '<v_nm>' '=' '<expr>' $action2 ) ; (
'<a_nm>'
        $action3 '(' [ ( ( '<v_nm>' ; '<val>' ; '<tag>' )
        $attr3 )#( ',' $comma1 ) ] ')' $comma2 ) )#( ',' )
.
   S  :  (  '<u_nm>'  ;  '<w_nm>'  ;  '<d_nm>'  ;  '<g_nm>'  ;
'exit') $state3 .
```

Для проверки правильности записи регулярных выражений использовалось инструментальное средство SynGT (Syntax Graph Transformations) [6; 7]. На рис. 1 показано графическое представление правил грамматики языка CIAO.



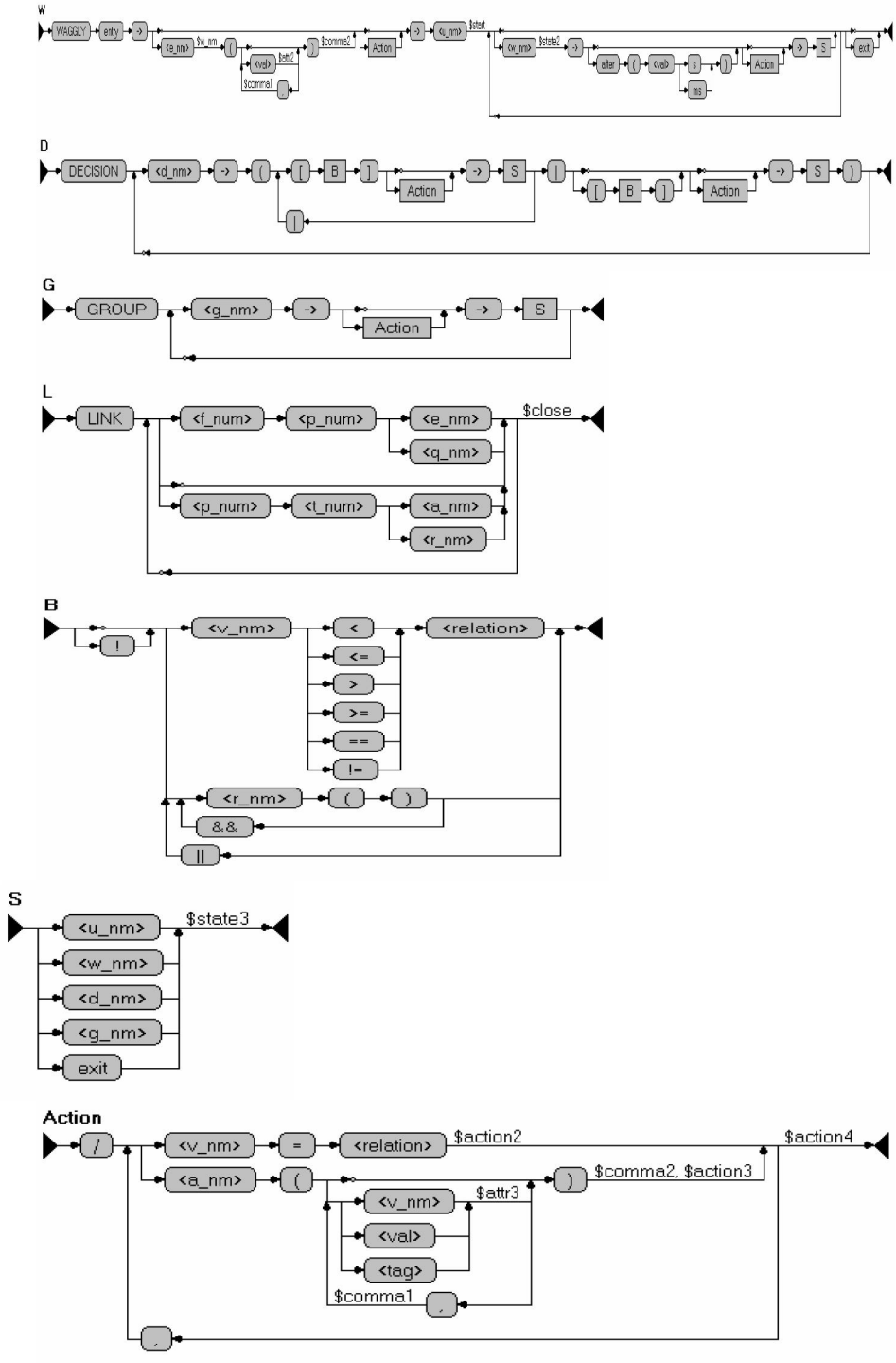


Рис. 1. Синтаксические графы для нетерминалов грамматики

Алгоритм метода создания реагирующей системы

Для проектирования и реализации параллельных реагирующих систем предлагается метод ReSyD (Reactive Systems Design), в основу которого положена модель взаимодействующих автоматных объектов (рис. 2). Алгоритм метода создания реагирующей системы состоит из семи шагов:

(1) проводится анализ предметной области, строятся диаграммы использования с учетом функциональных требований к системе; **(2)** выделяются сущности со сложным поведением (автоматные объекты), составляется список команд и запросов, параллельные операции назначаются разным объектам; **(3)** определяются состояния и переходы между ними для каждого объекта, строятся диаграммы состояний; **(4)** по списку команд и запросов устанавливаются связи между автоматными объектами, на схеме связей указываются контракты предоставляемых и требуемых интерфейсов; **(5)** анализируется схема связей: сильно связанные или сходные по функциональному назначению объекты группируются в компоненты, строится архитектура системы в виде диаграммы компонентов; **(6)** по диаграммам состояний определяются классы программных объектов и классы интерфейсов; функции и переменные, декларированные на диаграммах, имплементируются в программный код на целевом языке; **(7)** в случае необходимости код дополняется деталями, не отображенными на диаграммах.

Рис. 2. Алгоритм метода *ReSyD*

В качестве примера описания метода выбрана система управления работой лифта. Рассмотрим текстовую постановку задачи в несколько упрощенной форме по сравнению с фундаментальной книгой [9].

Простой лифт в многоэтажном доме должен перевозить пассажиров с этажа на этаж. Лифт имеет кнопки вызова на каждом этаже и кнопки отправления на этажи в кабине. Система управления лифтом не принимает новых вызовов, пока не закончит обслуживание принятого вызова. Двери шахты и двери лифта открываются и закрываются системой управления, причем падение пассажиров в шахту лифта через несвоевременно открытые двери шахты или лифта должно быть исключено. В пол лифта встроен датчик давления. Система управления должна исключить движение с перегрузкой или недогрузкой лифта. В кабине лифта имеется освещение. Система управления должна включать и выключать освещение так, чтобы экономить электричество, но не пугать пассажиров несвоевременным выключением света.

Вначале по словесному описанию строится граф переходов состояний, отражающий поведение лифта (рис. 3). Методика явного выделения состояний для краткости опускается, в основных чертах она следует парадигме автоматного программирования [10].

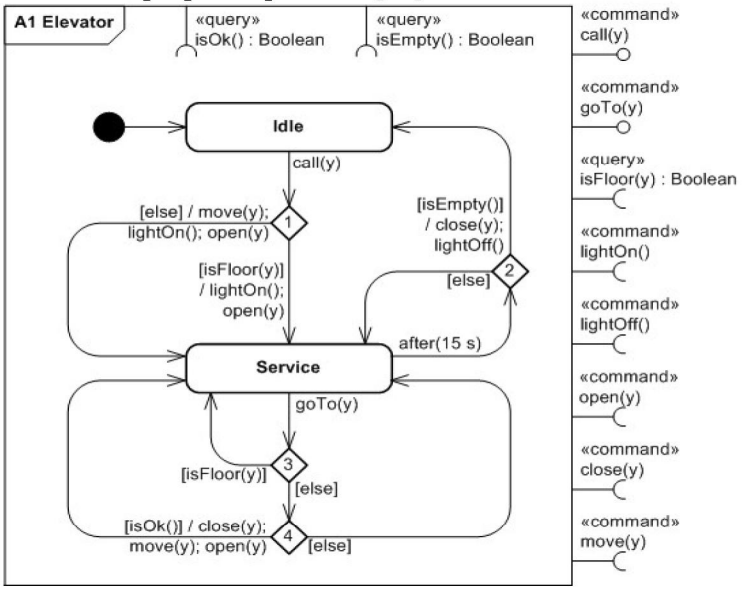


Рис. 3. Диаграмма состояний автоматного объекта Elevator

Графической диаграмме переходов состояний соответствует текстовое представление на языке CIAO (листинг 1).

```
#1 Elevator
EVENT
call(y : Integer)
goTo(y : Integer)
ACTION
lightOn()
lightOff()
open(y : Integer)
close(y : Integer)
move(y : Integer)
REQUEST
isFloor(y) : Boolean
isOk() : Boolean
isEmpty() : Boolean
STATE
idle -> call(y) -> dcsn1
service -> after(15 s) -> dcsn2
service -> goTo(y) -> dcsn3
WAGGLY
entry -> -> idle
DECISION
dcsn1 -> ( [isFloor(y)] /
   lightOn(), open(y) -> service
   | / move(y), lightOn(), open(y
   ) -> service )
dcsn2 -> ( [isEmpty()] / close(y
   ), lightOff() -> idle | ->
   service )
dcsn3 -> ( [isFloor(y)] ->
   service | -> dcsn4 )
dcsn4 -> ( [isOk()] / close(y),
   move(y), open(y) -> service |
   -> service )
LINK
#2 #1 call
#2 #1 goTo
#1 #2 isFloor
#1 #2 lightOn
#1 #2 lightOff
#1 #2 open
#1 #2 close
#1 #2 move
```

Листинг 1. Текстовое представление поведения автоматного объекта Elevator на языке CIAO

После применения кодогенератора SynGT к тексту в листинге 1 получена заготовка текста для языка C++ (листинг 2).

```
typedef enum {
  idle, service
} State1;

class I1
{
public:
  void call(int y);
  void goTo(int y);
};

class A1 : public I1
{
private:
  State1 y_;
  I2 *out_;
  bool isOk();
  bool isEmpty();

public:
  void run() {
    y_=idle;
    int y;
    while (1) {
      event = wait(1);
      switch (y_) {
        case idle:
        {
          if (event==call(y)) {
            if (isFloor(y)) {
              out_->lightOn();
              out_->open(y);
              y_=service;
            }
            else {
              out_->move(y);
              out_->lightOn();
              out_->open(y);
              y_=service;
            }
          }
          break;
        }
        case service:
        {
          if (event==goTo(y)) {
            if (isFloor(y)) {
              y_=service;
            }
            else {
              if (isOk()) {
                out_->close(y);
                out_->move(y);
                out_->open(y);
                y_=service;
              }
              else {
                y_=service;
              }
            }
          }
          if (event==timer
             (15000)) {
            if (isEmpty()) {
              out_->close(y);
              out_->lightOff();
              y_=idle;
            }
            else {
              y_=service;
            }
          }
          break;
        }
      }
    }
  }
};
```

Листинг 2. Программа на языке C++ для автоматного объекта Elevator

Отличительным свойством данной методики является то обстоятельство, что полученное решение является доказательно верным. Действительно, требование безопасности «исключить падение пассажиров в шахту через несвоевременно открытые двери» выполнено, поскольку из диаграммы на рис. 2 видно, что действие open(y) выполняется в любом случае только при выполнении предусловия isFloor(y). Все дальнейшие преобразования сохраняют это и другие свойства диаграммы переходов состояний.

Заключение

Описаны структура и текущее состояние грамматики языка CIAO для компиляции и проведения экспериментов с программами на языке CIAO для моделирования поведения сетей взаимодействующих автоматных объектов. Особенностью данной разработки является использование языка C++ с набором инструментальных средств для создания и отладки всех компонентов данной программы.

Дальнейшая работа будет состоять в расширении функциональности созданных компонентов (прежде всего кодогенератора и программы имитационного моделирования сгенерированного кода), в уточнении грамматики и семантики входного языка CIAO и в проведении экспериментов по программированию и исполнению реальных приложений.

Построенная автоматная модель таких расширенных диаграмм реализована в языке CIAO и обеспечивает модульность описания поведения программной системы, а также задает интерфейс взаимодействия как автоматов, так и любых других программных компонентов. Язык CIAO превосходит известные аналоги в части предоставления механизмов кооперативного взаимодействия, асинхронности и параллельности для широкого класса реагирующих систем.

Литература

1. Attribute-Based Approach of Defining the Secure Behavior of Automata Objects / F. A. Novikov [et al.] // Proceedings of SIN 2017 conference (SIN 2017). NY., 2017. P. 67–72. DOI: https://doi.org/10.1145/3136825.3136887

2. Афанасьева И. В., Новиков Ф. А. Архитектура программного обеспечения систем оптической регистрации // Информационно-управляющие системы. 2016. № 3. С. 51–63. DOI: 10.15217/issue 1684-8853.2016.3.51

3. Новиков Ф. А., Афанасьева И. В. Кооперативное взаимодействие автоматных объектов // Информационно-управляющие системы. 2016. № 6. С. 50–63. DOI: 10.15217/issn1684-8853.2016.6.50

4. Harel D. Statecharts: a Visual Formalism for Complex Systems // Science of Computer Programming. 1987. V. 8. P. 231–274.

5. Шалыто А. А. Парадигма автоматного программирования // Научно-технический вестник СПбГУ ИТМО. 2008. Вып. 53. С. 3–24.

6. Fedorchenko L. Regularization of Context-Free Grammars. Saarbrucken: LAP LAMBERT Academic Publishing, 2011. 180 p.

7. Fedorchenko L., Baranov S. Equivalent Transformations and Regularization in Context–Free Grammars // Cybernetics and Information Technologies (CIT). Sofia, 2015. V. 14, No. 4. P. 11–28.

8. Aho A., Sethi R., Ullman J. Compilers: Principles, Techniques and Tools. Addison-Wesley, 1986. 796 p.

9. Кнут Д. Искусство программирования для ЭВМ. Т.1 Основные алгоритмы: пер. с англ. М.: Мир, 1976. 736 с.

10.Поликарпова Н. И., Шалыто А. А. Автоматное программирование. СПб.: Питер, 2011. 176 с.

A METHOD FOR DESCRIBING SYSTEMS WITH COMPLEX BEHAVIOR BASED ON THE PRINCIPLES OF GENERALIZED AUTOMATA

Ludmila N. Fedorchenko
Cand. Sci. (Engineering), Senior Researcher,
St. Petersburg Institute for Informatics and Automation RAS (SPIIRAS)
39, 14th Line of V. O., St. Petersburg 199178, Russia
E-mail: fedorchenko2010@gmail.com

Irina V. Afanasyeva
Leading Engineer,
Special Astrophysical Observatory RAS (SAO RAS)
Nizhny Arkhyz 396167, Russia
E-mail: riv615@yandex.ru

Automata-based models are most convenient for designing systems with complex behavior, but they have a number of limitations, such as the lack of support for asynchrony and concurrency. Special language tools, both graphic and textual are often used to describe such models. The article presents a method for describing systems with complex behavior using the high-level programming language CIAO (Cooperative Interaction of Automata Objects). The program of this language generates a system of automatic machines, which simulate the functioning of the initial reactive system.

Keywords: asynchronous parallel reactive systems; state-transition graph; syntactic flowgraph in regular form.

References

1. Novikov F. A., Fedorchenko L. N., Vorobiev V. I., Fatkieva R. R., and Levonevskiy D. K. Attribute-Based Approach of Defining the Secure Behavior of Automata Objects. Proceedings of the 10th International Conference on Security of Information and Networks (SIN 2017), J. B. Sartor, Th. D'Hondt, and W. De Meuter (Eds.). New York, USA: ACM, 2017. Pp. 67–72 DOI: https://doi.org/10.1145/3136825.3136887

2. Afanasyeva I. V., Novikov F. A. Arkhitektura programmnogo obespecheniya sistem opticheskoi registratsii [Software Architecture of Optical Registration Systems]. *Informatsionno-upravlyayushchie sistemy — Information Management Systems*. 2016. No. 3. Pp. 51–63. DOI: 10.15217 / issue 1684-8853.2016.3.51

3. Novikov F. A., Afanasyeva I. V. Kooperativnoe vzaimodeistvie avtomatnykh ob"ektov [Cooperative Interaction of Automaton Objects]. *Informatsionno-upravlyayushchie sistemy — Information Management Systems*. 2016. No. 6. Pp. 50–63. DOI: 10.15217 / issn1684-8853.2016.6.50

4. Harel D. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*. 1987. V. 8. Pp. 231–274.

5. Shalyto A. A. Paradigma avtomatnogo programmirovaniya [Paradigm of Automata-Based Programming]. *Nauchno-tekhnicheskii vestnik Sankt-Peterburgskogo gosudarstvennogo universiteta informatsionnykh tekhnologii, mekhaniki i optiki — Scientific and Technical Bulletin of St. Petersburg State University of Information Technologies, Mechanics and Optics*. 2008. V. 53. Pp. 3–24.

6. Fedorchenko L. *Regularization of Context-Free Grammars*. Saarbrucken: Lap Lambert Academic Publ., 2011. 180 p.

7. Fedorchenko L., and Baranov S. Equivalent Transformations and Regularization in Context-Free Grammars. *Cybernetics and Information Technologies (CIT)*. 2015. V. 14. No 4. Pp.11–28.

8. Aho A., Sethi R., Ullman J. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986. 796 p.

9. Knuth D. *The Art of Computer Programming*. V. 1. Fundamental Algorithms. Massachussets, USA: Addison-Wesley, 1968. 634 p.

10. Polikarpova N. I., Shalyto A. A. *Avtomatnoe programmirovanie* [Automata-Based Programming]. St Petersburg: Piter Publ., 2011. 176 p.