computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific Journal of Computing

ОБЪЕКТНОЕ ПРЕДСТАВЛЕНИЕ КОНТУРА УПРАВЛЕНИЯ ДИСКРЕТНЫМИ ПРОЦЕССАМИ НА ОСНОВЕ СИСТЕМ ПРОДУКЦИЙ

Тихомирова Е.В.

Белорусский государственный университет
информатики и радиоэлектроники,
г. Минск

Резюме: *Рассматривается задача программной реализации контура управления дискретными процессами в системах организационно-технологического уровня с применением технологии объектно-ориентированного моделирования. В качестве базового формализма при решении задачи используются системы продукций. Вопросы интерпретации системы продукции решаются по схеме интерпретации процессов на сетевой структуре.*

Ключевые слова: *логическое управление, дискретные процессы, сети Петри, системы продукций.*

ВВЕДЕНИЕ

Возросший повсеместно интерес к проблемам построения высокоэффективных и высоконадежных систем автоматизированного управления, например в промышленности, транспорте, энергетике и т.д., способствовал широкому применению методов моделирования для исследования функционирования и поведения сложных объектов [1]. В общем виде процесс взаимодействия управляющей части с объектом управления и внешней средой может быть охарактеризован системой функциональных зависимостей *F:*

$$Y=F(X),$$

где $X=\{x_1,x_2,\ldots x_n\}$- входные данные, представляющие собой контролируемые состояния объекта управления или внешней среды;

$Y=\{y_1,y_2,\ldots,y_m\}$ – выходные данные, например, управляющие воздействия на объект управления.

Функционирование системы заключается в регулярном фиксировании фактов изменения переменных состояний системы с последующим рекуррентным изменением значений функционально – зависящих от них объектов представления выходных данных. Известны способы формализации таких систем, например автоматные модели, графы операций [2, 3, 4], однако способ их дальнейшего представления в процедурном виде не обсуждался.

Задача разработки способа представления в памяти ЭВМ зависимостей, отражающих процесс функционирования системы и проведения анализа этих зависимости в ускоренном и реальном режиме времени здесь будет рассматриваться с позиции технологии объектно-ориентированного моделирования.

1. ДИСКРЕТИЗАЦИЯ ЗАКОНОВ УПРАВЛЕНИЯ В ТЕРМИНАХ СЕТЕВЫХ МОДЕЛЕЙ

Пусть система характеризуется тройкой:

$$S=<V,P,A>,$$

где V – множество переменных состояния системы $(V_i\in V)$;

P – множество условий на V, задаваемых вычислимыми предикатами;

A – множество действий, соответствующих условиям P.

Каждое действие A – множество преобразований вида $V:=A(V)$, причем правые части этих преобразований обязательно вычислимы на V, а все члены правой части подвергаются преобразованию одновременно.

Любой элемент V_i из множества переменных состояний системы V, может быть охарактеризован четверкой функциональных зависимостей

$$V_i=(E_i, C_i, F_i, D_i),$$

где E_i– условия активизации элемента $E_i\in P$;

C_i – действия, вызываемые в системе при входе элемента в активное состояние, $C_i\in A$;

F_i – действие перехода при выходе элемента из активного состояния $F_i\in A$;

D_i – длительность во времени активной фазы элемента.

Процесс функционирования системы представляет собой регулярное применение процедур реализации элементарных действий из множества C (рис.1).

Рис.1. Графическое представление процесса функционирования системы.

Подходящим методом математического описания системы для последующего представления в ЭВМ является система продукций, представляющая собой набор продукционных правил:

ЕСЛИ <условие> ТО <действие>.

Система продукционных правил описывает множество объектов моделируемой системы и отношения между ними. Такое представление моделируемой системы удобно тем, что позволяет:

- однозначно фиксировать объекты и их свойства, действия над ними;
- специалисту (конструктору, технологу) сформулировать правила решений (поведения системы) в привычном для него представлении, т. е. в форме, близкой к справочникам, стандартам и другим источникам, что позволяет в дальнейшем избежать сложности формализации поведения объектов системы;
- модульность продукций дает возможность расширения и корректировки данной среды с учетом конкретной специфики.

Однако, формальное представление функционирования системы в виде продукционных правил не отражает динамики связи переменных состояний системы во времени. Нетрудно заметить, что задача интерпретации системы продукции – получение последовательности действий может решаться по схеме интерпретации процессов на временной сети Петри (*TPN*) в виде последовательности активизации переходов [5, 6]. Например, рассмотрим сеть Петри (рис.2)

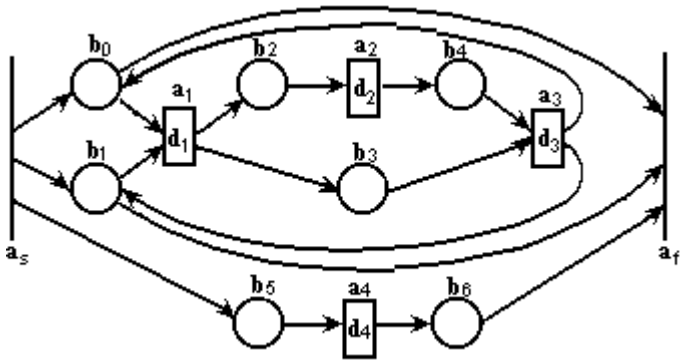

Рис.2. Графическое представление сети Петри

Введем обозначения:

$b_1, b_2, \ldots, b_n$ – переменные состояния $b_i \in B$,

$a_1, a_2, \ldots, a_m$- действия $a_i \in A$,

E, C и F – функции связывающие разметку позиций с весами дуг сети.

Сеть *TPN* может быть представлена структурой смежности:

выходные связи позиций:

$b_0 : a_f , a_1$
$b_1 : a_f ; a_1$
$b_2 : a_2$
$b_3 : a_3$
$b_4 : a_3$
$b_5 : a_4$
$b_6 : a_f$

выходные связи переходов:

$a_s : b_0, b_1, b_5$
$a_1 : b_2, b_3$
$a_2 : b_4$
$a_3 : b_0 , b_1$
$a_4 : b_6$

В случае *TPN* позиции соответствуют переменным состояния, а переходы – действиям системы. Если некоторый переход a_i стал пассивным, то это порождает необходимость проверки активизации переходов $\{a_{i+1}\}$ множества функций E_{i+1} которые содержат переменные измененные функцией F_i. Если условие E_{i+1} истинно, то выполняются действия C_{i+1} после чего активизируется a_{i+1} и проверяется возможность активизации переходов, условия активизации которых определены и на переменных функции C_{i+1}.

Таким образом, порождение процесса активизации переходов основано на восприимчивости отдельных переходов системы продукций к изменению только локальных переменных состояния. Отсюда следует, что на переходах системы продукций можно формально построить сеть (рис.3), связывающие переходы с потенциальной возможностью активизации.

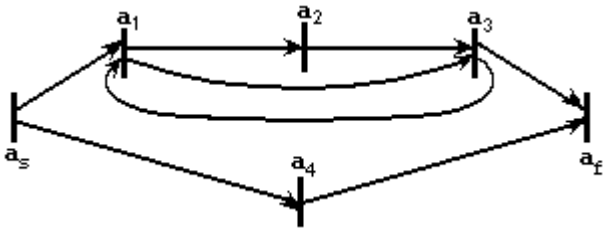

Рис. 3. Пример сетевого преобразования системы продукций.

Структура смежности связей переходов сети системы продукций (рис.3):

$a_s : a_1, a_4$
$a_1 : a_2, a_3$
$a_2 : a_3$
$a_3 : a_f, a_1$
$a_4 : a_f$

Определения функций для сетевого преобразования системы продукций :

$E(a_s)=0;$
$E(a_1)=b_0 \,\&\&\, b_1;$
$E(a_2)=b_2;$
$E(a_3)= b_3 \,\&\&\, b_4;$
$E(a_3)= b_5;$
$E(a_f)= b_0 \,\&\&\, b_1 \,\&\&\, b_6;$

$C(a_s)=0;$
$C(a_1)=b_0\text{--},b_1\text{--};$
$C(a_2)= b_2\text{--};$
$C(a_3)= b_3\text{--},b_4\text{--};$
$C(a_3)= b_5\text{--};$
$C(a_f)= b_0\text{--},b_1\text{--},b_6\text{--};$

$D(a_s)=0;$
$D(a_1)=d_1;$
$D(a_2)=d_2;$
$D(a_3)=d_3;$
$D(a_3)=d_4;$
$D(a_f)=0;$

$F(a_s)=b_0{+}{+},b_1{+}{+},b_5{+}{+};$
$F(a_1)=b_2{+}{+},b_3{+}{+};$
$F(a_2)=b_4{+}{+};$
$F(a_3)=b_0{+}{+},b_1{+}{+};$
$F(a_3)=b_6{+}{+};$
$F(a_f)=0;$

Таким образом, система продукции может быть представлена массивом функции:

$\{ a_i : < E_i, C_i, F_i, D_i >, i=1, ..., m\}$

и так называемой интерпретируемой сетью переходов. Такую структуру можно интерпретировать как переходную систему.

2. АЛГОРИТМИЧЕСКАЯ ИНТЕРПРЕТАЦИЯ СЕТЕВЫХ МОДЕЛЕЙ СРЕДСТВАМИ ЯЗЫКА С++

Из удобства содержательной интерпретации поставим в соответствие каждому элементарному действию на сети идентифицированный одношаговый процесс во времени – переход, характеризуемый четверкой функциональных зависимостей общего вида – a_i:< E_i, C_i, F_i, D_i>:

```
class Transition{
public:
  int d_time;                    // Временная задержка перехода
  Production<Transition>* prod;
  virtual int tr_E(void)=0;    // Схема процедуры вычисления условий
                               // активизации перехода
  virtual int tr_C(void)=0;    // Схема процедуры выполнения действия перехода
                               // при входе в активное состояние
  virtual int tr_F(void)=0;    // Схема процедуры выполнения действия перехода
                               // при выходе из активного состояния
  virtual int tr_D(void)=0;    // Схема процедуры вычисления длительности
                               // активной фазы перехода
  Transition(){}
  virtual ~Transition(){}
};
```

Условия активизации перехода могут определяться константой, логическим выражением или обращением к функции. Длительность задержки переходов в активизированном состоянии допускается задавать константой, переменной либо выражением. Функции и действия переходов могут включать любые операторы языка С++ и в частности *E* и *C* могут включать операторы деклараций объектов. Интервал активности перехода ограничен моментами вызова процедур *E* и *C* и по определению перехода оперативно привязан к временной оси. Однако не существует препятствий для изменения плана завершения активной фазы перехода. Подобные действия может выполнять другой переход, который скорректирует список событий на интерпретируемой сети.

Статическое описание сети представлено параметризованной структурой Frame, содержащей определение схемы связей переходов и позиций:

```
template<class transition>
struct Frame{
  int* Ia;          // Индексы списка связанных позиций переходов
  int* Ja;          // Связи переходов
  int* Ib;          // Индексы списка связанных переходов позиций
  int* Jb;          // Связи позиций
  transition** Tr; // Переходы сети
  int  n_tran;      // Количество переходов
  int  n_pos;       // Количество позиций
};
```

Динамическое описание сети осуществляется в классе `TPN`:

```
template<class transition>
class TPN:public Frame<transition>{
public:
  int* N;           // Поля ссылок списка событий
  int* F;           // Моменты времени пассивизации переходов
// Процедура интерптетации процессов
  void interptetator(){
    cout<<"Старт моделирования в момент времени t= "<<*F<<endl;
    Tr[*N]->tr_C();
    do{
// Обработка последовательности событий
      int t=*N;
      *N=N[t]; // Выборка очередного перехода
```

```
      *F=F[t]; // Момент пассивизации
       if (t) N[t]=n_tran; // Фиксация пассивности перехода t
// Обработка последствий пассивизации перехода t
      Tr[t]->tr_F(); // Выходные действия
      for (int i=Ia[t], j=Ia[t+1]; i<j; i++)
        for (int k=Ja[i], l=Ib[k], m=Ib[k+1]; l<m; l++) {
// Обработка последствий изменения разметки позиции k
          int n=Jb[l],f;
          if (N[n]!=n_tran) continue; // Отказ от активизации активного перехода
          if (Tr[n]->tr_E()) { // Проверка условий активизации
            Tr[n]->tr_C(); // Входные действия
// Планирование момента пассивизации перехода n
            f=F[n]=*F+(Tr[n]->tr_D());
            cout<<" t="<<F[n]<<endl;
            for (int p=0, q=*N; (q>0)&&(f>=F[q]); p=q, q=N[p]);
            N[n]=q, N[p]=n;
          }
        }
    }while(*N>0);
    cout<<"Финиш моделирования в момент времени t= "<<*F<<endl;
  }
  TPN(){}          // Конструктор
  virtual ~TPN(){} // Деструктор
};
```

Класс интерпретации системы продукций :

```
template<class transition>
class Production:public TPN<transition> {
public:
  int b0,b1,b2,b3,b4,b5,b6;
  Production(const Frame<transition> frame){  //Конструктор
    b0=0,b1=0,b2=0,b3=0,b4=0,b5=0,b6=0;
    n_tran=frame.n_tran;
    n_pos =frame.n_pos;
    int n_Ja=frame.Ia [n_tran];
    int n_Jb=frame.Ib [n_pos];
    Tr = new transition* [n_tran];
    Ja = new int [n_Ja];
    Ia = new int [n_tran+1];
    Jb = new int [n_Jb];
    Ib = new int [n_pos+1];
    N  = new int [n_tran]; *N=0;
    F  = new int [n_tran]; *F=0;
    for(int i=0;i<n_pos+1;i++)Ib[i]=frame.Ib[i];
    for(i=0;i<n_tran+1;i++)   Ia[i]=frame.Ia[i];
    for(i=0;i<n_Ja;i++)       Ja[i]=frame.Ja[i];
    for(i=0;i<n_Jb;i++)       Jb[i]=frame.Jb[i];
    for(i=0;i<n_tran;i++)    {Tr[i]=frame.Tr[i]; Tr[i]->prod=this;}
    for (i=1;i<n_tran;N[i++]=n_tran);
  }
  ~Production(){
    for(int i=0;i<n_tran;i++) delete Tr[i];
    delete[] Tr;
    delete[] Ia; delete[] Ja;
    delete[] Ib; delete[] Jb;
    delete[] N;  delete[] F;
  }
};
```

Определим классы переходов сети системы продукций (рис.3):

```
class Transition_as:public Transition{
public:
  virtual int tr_E(void){ return 0;}
```

```
  virtual int tr_C(void){
    cout<<"- as "<<endl;
    prod->b0++, prod->b1++, prod->b5++;
    return 1;
  }
  virtual int tr_F(void){ return 0;}
  virtual int tr_D(void){ return d_time;}
};

class Transition_af:public Transition{
public:
  virtual int tr_E(void){ return(prod->b0&&prod->b1&&prod->b6);}
  virtual int tr_C(void){
  cout<<"+ af";
  prod->b0--, prod->b1--,prod->b6--;
  return 1;
}
  virtual int tr_F(void){ return 0;}
  virtual int tr_D(void){ return d_time;}
};

class Transition_a1:public Transition{
public:
  virtual int tr_E(void){ return(prod->b0&&prod->b1);}
  virtual int tr_C(void){ prod->b0--, prod->b1--; cout<<"+ a1 "; return 1;}
  virtual int tr_F(void){ prod->b2++, prod->b3++; cout<<"- a1"<<endl; return 1;}
  virtual int tr_D(void){ return d_time;}
};

class Transition_a2:public Transition{
public:
  virtual int tr_E(void){ return prod->b2; }
  virtual int tr_C(void){ prod->b2--; cout<<"+ a2 "; return 1;}
  virtual int tr_F(void){ prod->b4++; cout<<"- a2"<<endl; return 1;}
  virtual int tr_D(void){ return d_time;}
};

class Transition_a3:public Transition{
public:
  virtual int tr_E(void){ return(prod->b3&&prod->b4); }
  virtual int tr_C(void){ prod->b3--, prod->b4--;cout<<"+ a3 "; return 1;}
  virtual int tr_F(void){ prod->b0++, prod->b1++; cout<<"- a3"<<endl; return 1;}
  virtual int tr_D(void){ return d_time;}
};

class Transition_a4:public Transition{
public:
  virtual int tr_E(void){ return prod->b5; }
  virtual int tr_C(void){ prod->b5--;cout<<"+ a4 "; return 1;}
  virtual int tr_F(void){ prod->b6++;cout<<"- a4"<<endl; return 1;}
  virtual int tr_D(void){ return d_time;}
};
```

Фрагмент программы использования разработанных классов:

```
void main(void) {
//...
// Схема связей переходов
  // Индексы списка связанных позиций переходов
  int Iax[] = {0,7,10,14,16,20,22};
  //Связи переходов
  int Jax[] = {0,1,2,3,4,5,6, 0,1,6, 0,1,2,3, 2,4, 3,4,0,1, 5,6};
```

```
            // as             af     a1       a2   a3       a4
// Временные задержки переходов
  int Dx [] = {0, 0, 2, 3, 4, 10};
            // as af a1 a2 a3 a4
// Схема связей позиций
  //Индексы списка связанных переходов позиций
  int Ibx[] = {0,2,4,5,6,7,8,9};
  //Связи позиций
  int Jbx[] = {1,2, 1,2, 3, 4, 4, 5, 1};
            // b0   b1   b2 b3 b4 b5 b6
  Frame<Transition>* frame=new Frame<Transition>;
  frame->n_tran=sizeof(Dx)/sizeof(*Dx);
  frame->n_pos=sizeof(Ibx)-1;
  int n_Ja=sizeof(Jax)/sizeof(*Jax);
  int n_Ia=sizeof(Iax)/sizeof(*Iax);
  int n_Jb=sizeof(Jbx)/sizeof(*Jbx);
  int n_Ib=sizeof(Ibx)/sizeof(*Ibx);
  frame->Tr=new Transition*[frame->n_tran];
  frame->Tr[0]=new Transition_as;
  frame->Tr[1]=new Transition_af;
  frame->Tr[2]=new Transition_a1;
  frame->Tr[3]=new Transition_a2;
  frame->Tr[4]=new Transition_a3;
  frame->Tr[5]=new Transition_a4;
  for(int i=0;i<(frame->n_tran);i++) frame->Tr[i]->d_time=Dx[i];
  frame->Ja=new int [n_Ja];
  frame->Ia=new int [n_Ia];
  frame->Jb=new int [n_Jb];
  frame->Ib=new int [n_Ib];

  memcpy(frame->Jb,Jbx,sizeof(Jbx));
  memcpy(frame->Ib,Ibx,sizeof(Ibx));
  memcpy(frame->Ja,Jax,sizeof(Jax));
  memcpy(frame->Ia,Iax,sizeof(Iax));

  Production<Transition>* production=new Production<Transition>(*frame);
  production->interptetator();  // Интерпретация процессов на сети

  delete frame;
  delete production;
//...
};
```

Результаты работы имитационной программы:

```
Старт моделирования процесса на TPN в момент времени t= 0
      - as
      + a1  t=2
      + a4  t=10
      - a1
      + a2  t=5
      - a2
      + a3  t=9
      - a3
      + a1  t=11
      - a4
      - a1
      + a2  t=14
      - a2
      + a3  t=18
      - a3
      + af  t=18
Финиш моделирования в момент времени t= 18
```

Таким образом, полученные результаты полностью соответствуют результатам интерпретации представленной рассматриваемой системой продукций *TPN*.

ЗАКЛЮЧЕНИЕ

Рассмотренный способ представления систем с дискретным характером поведения позволяет автоматизировать переход от их формального описания к программной реализации контура управления.

Предложенный алгоритм моделирования процессов на сетевых моделях допускает оценку эффективности ядра контура управления по критерию "память-быстродействие", что может быть использовано для конструктивного решения задач компенсации погрешностей воспроизведения законов управления в реальном времени.

ЛИТЕРАТУРА

1. *Бусленко Н.П. Моделирование сложных систем. – М.: Наука, 1978. – 400 с.*
2. *Малышев Н.Г. Структурно-автоматные модели технических систем. М.:Радио и связь, 1986. – 168с.*
3. *Юдицкий С.А., Магергут В.З. Логическое управление дискретными процессами. – М.: Машиностроение, 1987. – 175 с.*
4. *Питерсон Дж. Теория сетей Петри и моделирование систем. М.: Мир, 1984. – 264 с.*
5. *Ревотюк М.П. Моделирование и оптимизация управления дискретными процессами на основе интерпретируемых сетей. Диссертация на соискание ученой степени кандидата технических наук. – Мн.: МРТИ, 1989, – 120с.*
6. *Тихомирова Е.В. Объектная интерпретация систем продукций средствами языка C++ //Радиомир. Ваш компьютер, 2000.- №11.- С. 32-35.*

Екатерина Тихомирова *работает системным программистом "Way-Systems Inc.", Medford USA.*

Кандидат технических наук с 2002г по специальности 05.13.01 – Системный анализ, управление и обработка информации (промышленность).

Область научных интересов: объектно-ориентированное моделирование и проектирование систем управления, методы спецификация контура управления на основе переходных систем, алгоритмы интерпретации дискретных процессов на сетевых структурах, криптографические схемы и алгоритмы, Smart card технологии, архитектуры и модели цифровых сотовых телекоммуникационных систем, методы и алгоритмы разработки протоколов связи для беспроводных мобильных устройств.