The application of transition diagrams in data handling processes modeling

Magariu Nicolae
Moldova State University
magariu@usm.md

Abstract. **The method of data handling processes modeling by means of transition diagrams and transition diagram systems is considered. As well as the method of program specification construction on base of dynamic model representation described as a transition diagram or transition diagram system is analyzed. The structure of applications framework elaborated by means of transition diagram systems is proposed.**

Index Terms**: transition diagrams, transition diagram systems, functional modeling of data handling processes, applications framework**

I. THE PROBLEM CONTEXT AND THE PROBLEM

The process of applications development depends on the complexity of the problem solved by this application and it includes obligatory the phase of their modeling-designing [1, 2]. To get a preliminary estimate of the complexity of the technology that can be applied in concrete imperative Software Product (SP) development a problem classification depending on three main problem characteristics V, S, and E has been proposed [3]:

V - the volume of the data that would be processed by future SP; S - the specification level of domain data handling processes; E - the efficiency of the technological model applicable on corresponding SP development process.

Intuitively, the value diapasons have proposed for each characteristic V, S and E.

The proposed diapasons for V characteristic: V1 – less then 100 MB, V2 – from 100 MB up to 1GB, V3 - more then 1 GB.

The proposed diapasons for S characteristic: S1 – a data handling processes have specified with the help of algorithms, S2 – a processes are not specified with the help of algorithms but there are mathematical models that can be applied to specify domain data handling processes, S3 – a processes are not specified with the help of algorithms and the formal models that can be applied to specify domain data handling processes do not exist.

The proposed diapasons for E characteristic: E1 – technological models based on using the specialized tools, for example EXCEL, 1C, SGBD, etc., E2 – technological models based on using structural and/or object-oriented programming systems, E3 – technological models based on using of generic programming principles and CASE systems.

Arranging these values in three-dimensional table we obtain 27 classes of problems which are shown in Table 1. In every class of problems the families of similar problems can be pointed out. These problems can be solved with the help of one typical technological model.

So, after formulating the problem, the class and the family of the problem must be determined, and the possibility of applying corresponding technological model must be studied.

Table 1. Problems classes

E_1:

V_1S_1	V_1S_2	V_1S_3
V_2S_1	V_2S_2	V_2S_3
V_3S_1	V_3S_2	V_3S_3

E_2:

V_1S_1	V_1S_2	V_1S_3
V_2S_1	V_2S_2	V_2S_3
V_2S_1	V_3S_2	V_3S_3

E_3:

V_1S_1	V_1S_2	V_1S_3
V_2S_1	V_2S_2	V_2S_3
V_2S_1	V_3S_2	V_3S_3

Within the structured programming paradigm, two functional methodologies of data handling processes modeling are used: DFD (Data Flow Diagram) and SADT (Structured Analysis and Design Technique) [2]. Frequently, within these methodologies the main attention is taken to static models of applications considering that their dynamic model must be elaborated on coding phase. But the dynamic model of application is very important on designing phase. The importance of dynamic model of the SP is determined by the following circumstances:

1) The dynamic model is represented graphically and it is studied, evaluated and modified by non-programmers.

2) Basing on dynamic model the corresponding SP specification is constructed.

The automatic conversion of SP dynamical model into program specification is a problem of great practical interest.

Within the structured programming paradigm the dynamic model of a SP can be represented in one of the following forms: structured algorithm scheme, multilevel State Transition Diagram (STD) proposed by Yourdon and Transition Diagram (TD) [2,4].

The representation of the dynamic model of SP as a structured algorithm diagram is not quite suitable in case of modeling a complex data handling processes which requires the frequently user's intervention.

The multilevel STDs are not described formal and one meets some difficulties on its converting into the program specification.

The application of TD for dynamic model representation of the SP was investigated insufficiently. Partially the simple TDs are used on modeling of the event oriented applications [2, 5].

II. THE CONCEPT OF DATA HANDLING PROCESSES DYNAMIC MODEL REPRESENTATION BY MEANS A TD

In the formal languages theory, Transition Diagrams (TDs) are used for Finite Automata (FA) graphical representation [6]. It is known that a Finite Automata (FA) is the formal machinery for sentences recognition/ accepting of the defined language L, as well as it is considered as a transducer [6]. Here we consider the using of TDs for data handling processes and SPs dynamic model representation.

We will treat as an event an activity of data handling which will be executed by SP. We will consider that every event ***e*** has a name which identifies this event and every event is being realized by a subprogram ***f***.

A TD is considered as a graph (V, U), where V is a set of vertices (nodes), and U – a set of oriented edges (arcs). This graph can be represented graphically.

A vertex ***v*** (***v***ϵV) is being represented graphically by a circle with a number inside. It represents an automata state of waiting events. The TD has one start vertex and it has a number 0 or 1. This start node represents start state of FA. A TD can have one or more final states. The final node is being represented graphically by double circle with a number inside.

An edge ***u*** (***u***ϵU) is an arrow (or arc) which connects two nodes. It can be defined as a pair of nodes. First node from this pair is the beginning of the arc, and second node – the end of the arc. An arc represents a transition from the beginning nod to the end node of the arc. It can be labeled with the name of the event. The label of the arc represents the event which has to be executed before transition.

A complete path of TD graph defines the events execution order of the defined list of events. Obviously that any complete path of TD graph must contain at least one labeled edge.

Obviously only deterministic TDs (or Deterministic FAs (DFAs)) can be used for dynamic modeling of SP.

2.1 Methodological aspects of TDs elaboration

The input alphabet of a DFA represented as a TD can be constructed during the specifying functional requirements to the SP. A functional requirement is described in more detailed way with the help of a scenario. The domain expert participates actively in description of scenario. On analyzing the scenario that corresponds to functional requirement the necessary activities (events) are distinguished and the functional requirement is specified as a list of event names. Events are registered in two-dimensional table (events dictionary), which contains three columns and multiple rows. The first column contains the event name, the second – short textual specification of the activity associated with event name, and the third column - prototypes of subprograms realizing corresponding activity (event).

While specifying functional requirements a two-dimensional table (requirements dictionary) is used. It includes three columns and multiple rows. The first column contains the requirement names; the second column - the list of events that specify corresponding requirement and the third column contain short textual specifications of the requirement.

Construction of the events and requirements dictionaries as well as the verification of their correctness is one of the most important phases of the TD construction process.

The general principle of TD construction consists in representation of one functional requirement as the unique complete path of TD. If lists of event names specifying different requirements have common parts then two or more complete paths of TD graph can coincide partially.

III. THE GENERAL DESCRIPTION OF THE ELABORATION PROCESS OF A SIMPLE SP

Input string of a deterministic DT is a list of event names which represents a specification of a functional requirement to the SP. A deterministic TD is built in such way that it accepts only input strings which represent functional requirements to the SP.

The general method of SP dynamic model elaboration as a TD includes following activities:

1. Elaboration and representation of each functional requirement as a list of events. (The set of all lists of events which represents functional requirements to the SP is considered as a language *L* of DFA)
2. Construction of the deterministic DT (or DFA) which accepts *L* language.

Let us assume that all functional requirements to the SP had represented as lists of events and the suitable deterministic TD had elaborated.

The algorithm of Functional Requirement Interpretation (FRI) is proposed.

3.1 The general description of the FRI algorithm

//Data description

S – Events queue.

Q **–** Variable, which keeps the number of current state (number of TD vertex) of the TD.

X – Variable, which keeps the event name read from S.

//Description of the used function

eread(X) function realizes the transferring the event name from S to X.

//The main part of the FRI algorithm

…

```
While (the node from Q is not final node)

   if (an arc leaving the current node and
      labeled with the event name from
      variable X exists)

      {Execute the subprogram, which
      corresponds to event name from
      variable X;

      Assign the number of arc end node to
      variable Q;

      eread(X)}
```

…

The *FRI* algorithm has the following important characteristics:

1) The algorithm work is controlled by TD.
2) The algorithm doesn't depend on TD.
3) The list of event names (or the prefix of this list) which specifies a functional requirement is an input parameter for the algorithm, and concrete elements of this list can be pointed at algorithm's run-time.

>From characteristics 1) - 3) it results that the algorithm *FRI* algorithm is invariant relative to any DT. But separate TD must be elaborated for every concrete problem.

The characteristic 3) demonstrates that users can control the computing process in the limits defined by TD.

Let's assume that the *FRI.C* function was constructed in C language. Then for constructing concrete SP one must elaborate dynamic model of this SP in the form of deterministic TD and represents it in acceptable form for *AFRI*.C function. In that case *AFRI*.C function can be considered as an applications framework.

IV. THE ASSOCIATION MODE OF TDs INTO TDS. CONSTRUCTION AND USAGE OFTDS

Let's assume that the TD has been constructed. Obviously, the FRI algorithm will function only if the subprograms realizing events have been already modeled and constructed. Some of these subprograms can be very complex. While modeling this subprograms one from three methods can bee used: modeling with the help of structural algorithm schemes, modeling with the help of multilevel state transition diagrams, and modeling with the help of TD. The selection of the dynamic model representation for corresponding subprogram depends on peculiarities of computing process being modeled. Thus, when the process is completely controlled by actors, modeling with the help of TD will be more convenient.

The case when a subprogram ***f*** realizing the event ***e*** can be modeled with the help of a TD is of great interest. Let's consider that there is an arc of the current TD labeled with event name ***e*** and the dynamic model of subprogram ***f*** is represented by new TD. While modeling the appropriate subprogram ***f*** by means of other TD, the current TD should be changed. Two variants of current TD changing should be considered in this case:

1) Substitution of the arc labeled with ***e*** name with the new TD which models the subprogram ***f***.
2) Changing the arc label ***e*** with the name of new TD which models the subprogram ***f***.

The substitution of the current TD's arc with the TD leads to the more complex current TD which can become very difficult for analyzing.

The second variant points out the formal way of TDs association and leads to representation of the application dynamic model as a TDS.

A TDS represents a TD set. Each TD from this set has unique name. Edges of each TD can be labeled by an event name or by a TD name from the set of TD names. The pointed out TD from the set of TDs is marked out as a main TD. The main TD represents the general dynamic model of the complex SP. Other TDs from the set represent the dynamic models of complex SP subsystems.

The representation of the complex SP dynamic model with the help of TDS is more convenient since each TD of the TDS can be elaborated and analyzed separately.

It must be mentioned that the TDS notion was proposed and used by M. Conway [7]. Conway had proposed to specify the syntax of formal languages by means of TDS and had elaborated the diagrammer (compiler) controlled by TDS. David Bruce Lomet had studied the Conway's diagrammer and had proved diagrammer's equivalence to a restricted Deterministic PushDown Acceptor (DPDA) called a nested DPDA [8]. He had established that the class of nested DPDA's is capable of accepting all deterministic context-free languages. The author studied Conway's TDS and diagrammer and elaborated a special class of TDS allowing the diagrammer to function in a deterministic way. This class of TDS was applied in the APL interpreter construction and for data processing modeling [9].

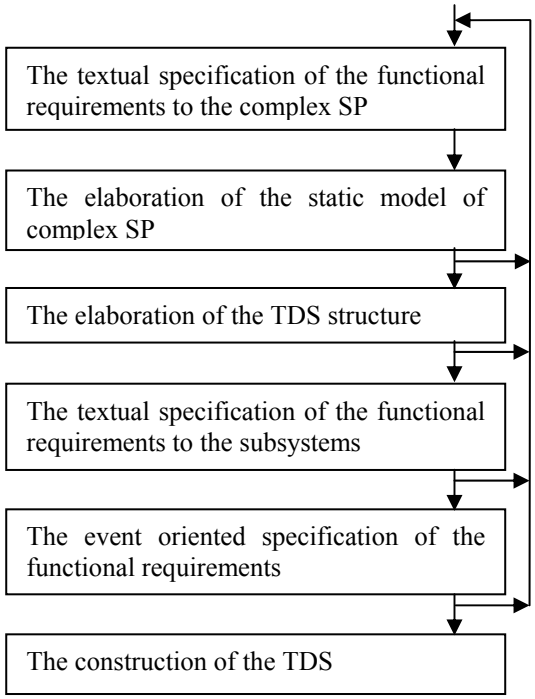

Fig.1 The model of TDS elaboration

The dynamic model representation of complex SP by means of TDS correlates very well with the top-down designing of complex SP. The dynamic model of complex SP can be represented by a TDS containing a main diagram named *SP_name* and TDs for all subsystems. The main diagram specifies the general behavior of the complex SP subsystems. It contains the arcs marked with the names of transition diagram from TDS. The TD of a subsystem specifies its behavior. A TD for each subsystem is elaborated applying the concept described in the compartment 2.1.

The general description of the processes of TDS construction is shown in the fig.1.

During TDS elaboration it is very important to ensure a deterministic transition from one state to another in each TD from the TDS. The constraints on TD structure providing deterministic transitions in each TD from a TDS were elaborated in [10].

The algorithm interpreting the functional requirement to the complex SP was constructed. It was named FRITDS and it is shown in Appendix A. Some important characteristics of the FRITDS algorithms:

1) The FRITDS algorithm is controlled by DTS.
2) The FRITDS algorithm covers the FRI algorithm.

V. THE GENERAL DESCRIPTION OF THE ELABORATION PROCESS OF COMPLEX SP

The complex SP elaboration process includes the following phases:

- The construction (in the programming language) of the AFRITDS function which corresponds to the algorithm AFRITDS;
- The construction of DTS representation accepted by AFRIS function.

It must be mentioned:

1) The AFRITDS function is constructed only one time for all SP.
2) A DTS is an input parameter for AFRIDS function.

The DTS must be constructed for each concrete problem. But the construction of TDS for the problem from a family of problems can be done by modifying the TDS of other problem from this family.

Conclusions

The stated results can serve as theoretical base for building the tool kit for automatic construction of complex SPs modeled with the help of TDS. TDS is a suitable method for modeling the service oriented SP.

APPENDIX A

The general description of the FRITDS algorithm

```
//Date description
   S – Events queue.

   X – The variable keeping the current
       event name.
   i – The variable keeping the number of
      current DT from TDS. The main TD of
      the TDS has the number 0.
   q - The variable keeping the current
      node number of current TD of TDS.
//General description of the algorithm
   steps
//The main diagram is fixed as the current
   TD
   ST1. i = 0;
//The initial nod of current TD is fixed.
   ST2. j = 0;
   ST3. Put the list of events names or the
        prefix of this list in events queue;
   ST4. eread(X);
   ST5. while (the current node is not the
                final node of the main
                diagram)
          {while (the current node is not the
                final node of the no main
                diagram)
             if(exist the arc marked with
                event name keeping in X)
               {Execute the subprogram
               corresponding to the event
               name from X;
               Execute the transition -change
               the value of variable j;
               eread(X);}
                else
                 if (exist the diagram
```

```
                accepting the event name
                keeping in X)
              {Memorize variables i and j
               in stack;
              Assign to variable i the
               number of selected TD;
               j=0;}
              else
               if(exist the non marked
                   arc)
            {Execute the transition on
             this arc by assigning to
             variable j the
             corresponding value}
ST6. Set up the values of variables i
     and j using values from the stack;
ST7. Execute the transition on the arc
     marked with the name of passed TD;
          }
```

REFERENCES

[1] Magariu N. Algorithmic description and programming. CEP USM, Chisinau, 2005. – 75 p. (in Romanian)

[2] G. S. Ivanova. Programming Technology. Moscow, Bauman N. E. PTR. 2002. - p. 320 (in Russian)

[3] Magariu N. "Some generalization about applications development process." The collection of scientific articles "Information Systems Design" of "Information Systems in Economy" and "Computing Systems and Programming" departments of State Engineering and Economic University of St. Petersburg. St. Petersburg, 2006. – pp. 145-152. (in Russian)

[4] http://yourdon.com/strucanalysis/wiki/index.php?title

=Chapter 13

[5] Stephen Ferg. Event-Driven Programming: Introduction, Tutorial, History. Version 0.2 – 2006-02-08. http://Tutorial_EventDrivenProgramming.sourceforge.net

[6] Hopcroft J., Motwani R., Ullman J. Introduction to Automata. Theory, Languages and Computation. MA: Addison-Wesley, 2001. —521 p.

[7] Melvin E. Conway. "Design of a separable transition-diagram compiler." Communications of the ACM, Volume 6, Issue 7 (July 1963). -pp. 396–408.

[8] David Bruce Lomet. "A Formalization of Transition Diagram Systems", Journal of the ACM (JACM) V. 20 , Issue 2 (April 1973). -pp. 235–257.

[9] Magariu N.A. "The application of transition diagrams on interactive programming system implementation." Mathematical researches of Moldova Science Academy, issue 107. The theory and practice of programming. Chisinau, Stiinta, 1989. -pp. 100–110. (In Russian)

[10] Magariu N. "The representation method of a complex software system dynamic project." Computer Science Journal of Moldova, V. 16, Nr. 2(47), Kishinev, 2008. – pp. 223-239.