Computer Science Journal of Moldova, vol.**16**, no.2(47), 2008

The representation method of a complex software system dynamic project

Nicolae Magariu

Abstract

The system of transition diagrams having the capacity to define the dynamic links between components of complex software system presented at different levels of specification is proposed. The model of complex software development based on the applying of transition diagrams systems is considered. The advantages of the model are analyzed.

Key words: software design, transition diagram, dynamic project of software system.

1 The problem

The necessity of intensification of human society economical development on the one hand, and advanced performances of modern computers on the other hand, force automation of the most complex Information Handling Processes (IHP) in the field of economics. This automation can be realized by means of Complex Information System (CIS) [1]. CIS development needs the big intellectual efforts of a group of developers in the course of several years. In the context of CIS development a special attention is given to CIS design phase, especially to CIS dynamic aspects design: components and subcomponents behavior specification, correlation of system's static and physic aspects with its dynamic aspects, etc. One of the well known means, which is applied to specify dynamic aspects of simple programs, is the transition diagram [2, 14]. An attempt to specify dynamic aspects of a CIS with the help of single transition diagram causes serious difficulties. Applying

a set of transition diagrams to specify CIS dynamic aspects can be a good solution of the posed problem. A model of applying the Systems of Transition Diagrams (STD) in CIS dynamic projects elaboration is proposed.

2 Preliminary

STD had been used for the first time in 1963 by American scientist Melvin E. Conway when he had developed a separable compiler [4]. Conway had specified the syntax of COBOL language utilizing STD, and had proposed an original algorithm of compilation which he had named diagrammer. A diagram of the STD has a double destination. It defines the syntax of language construction on the one hand and specifies the model of abstract automata for this construction implementation on the other hand.

As automata model a diagram consists of nodes and edges, where nodes represent states and edges define transitions from one state to another [2,5, 14]. A diagram can have one initial node (or state), one or more final nodes and no one or several intermediate nodes. An initial or intermediate state defines the opportunity to respond to defined events occurrence. A final state defines the passing interruption. The transition from one state to other is pointed by an edge. Edges can be marked by terminal symbols of the language or by name of a diagram. This method of interlinking between diagrams characterizes a Conway STD. A program unit can be associated with every edge. This unit is executed when the corresponding edge is passed. The STD includes the main diagram -"program" and several diagrams for presentation of all syntactic constructions of the language. The diagrammer is to pass full main diagram path having read the symbols of the program which are saved in input line. It passes the full path by steps. One step includes: - reading of one symbol from the input line and selection of the edge for transition to the other state; - execution of the program unit associated with the selected edge; - transition to the other state pointed by the selected edge. It finishes passing when the current state is a final state or the transition from current state isn't possible. It

starts the input line analysis from the initial state of the main diagram and recognizes program constructions in nondeterministic way - with returns in input line. Conway determines the possibility of STD and diagrammer application for other languages compilation as well. The STD can be constructed by using the formal description of the language syntax.

David Bruce Lomet had studied the Conway diagrammer and had proved diagrammer's equivalence to a restricted Deterministic Push-Down Acceptor (DPDA) called a nested DPDA [5]. He had established that the class of nested DPDA's is capable of accepting all deterministic context-free languages.

The author has been studied Conway's SDT and diagrammer independently of D. B. Lomet. There was elaborated a special class of STD which allows the diagrammer functions in deterministic way. This class of STD was applied in time of the APL interpreter construction [6].

APL is an interactive language and the syntax of its constructions is very simple [7]. User variables can obtain as a value a numerical array or an array of *char* type with arbitrary number of dimensions. APL operators can be nullary, unary or binary. Operands of the operators can be some expressions, evaluated to arrays, with arbitrary number of dimensions. Semantics of the APL operators is very consistent and it can be described in C language using tens or hundreds of instructions. Almost all operators in APL have equal priority (with small exceptions). The user can create and give to APL system an expression for execution or can create a User Defined Functions (UDF). A UDF prototype has the structure similar to the APL basic operator's structure. A UDF can be made up from expression instructions or branching instructions. Jumps can be made inside UDF only. A UDF can't be defined inside another UDF. All UDFs necessary for solving a problem or a family of problems are stored in a special Work Space (WS). An expression can contain invocations of UDF. To solve a problem by means of APL system, one needs to create the set of UDF in WS and give the APL expression to APL system. The execution of this expression leads to execution of UDFs, which are invoked directly or indirectly from this expression. So, the STD of the APL expression

defines the order of execution of a UDF from WS. One can say that the APL expression defines a control message to realize functionality, which is necessary for a user and the STD of APL expression defines the rules of execution of this expression. The diagrammer interprets the APL expression according to the rules defined by STD.

This context suggests the idea about using the STD in designing and construction of CIS [8]. In this case the process of data handling is considered as the execution of an actions sequence. One event provokes an action or an activity of data handling. A user function can be defined as a sequence of events. The STD defines the correctness of events sequence and the action of data handling which corresponds to each event of the sequence. Thus the STD represents the dynamical project of a CIS. An input line contains a sequence of events names – specification of the functional requirement of the user. The execution of actions of data handling which corresponds to this sequence of events has to give the user result. The diagrammer represents the model of interpretation control of actions corresponding to events pointed in an input line.

Ed Yourdon had researched a software system modeling by means of Transition Diagrams Systems of States (TDSS) [3]. He elaborated the method of creation of TDSS in the context of modeling the software systems oriented to data flaw. TDSS are the other form of dynamic project representation. The events and the program units aren't associated to edges and the links between diagrams aren't pointed evidently in TDSS.

3 A transition diagrams and STD as a dynamic projects

Two types of diagrams are used in automaton modeling: transition diagrams and state diagrams or machine [9]. Transition diagrams are used in modeling of G.H. Meally automaton and state diagrams are used in modeling of E.F Moore automaton.

The diagrams application in designing of automaton for different

objects control (for example, flying apparatus, seagoing ships, etc.) is studded in a number of publications [10,11]. The elaboration of such automatons begins with the specification of the object and their states. The object is described by means of finite set of variables. Every variable represents a property of the object. The value of each variable can be changed in time of object activity. An object state is specified by concrete values of a subset of variables. After this the automata model construction follows. This model is represented by means of one or several diagrams. For every state of the diagram there are specified: - the signals (or events names) which can be received from the object; - the procedure of control signals generation for every received event name; - the states in which the automata can be passed. If several diagrams are used under automata model construction then the association of these diagrams is defined. The method of diagram association is defined in particular way proceeding from feature of the concrete problem. As a rule such modeling is realized by means of state diagrams.

On the base of automata model the functional automata schema represented by means of logical operations or the program which describes the automata behavior is constructed.

The method of program construction on the base of the automata diagram is studied in the works [11,12]. In specification of such programs the separation of states describing from control process describing is proposed.

On the base of accumulated experience in construction of program for objects control the original style of programming was developed – Automaton Oriented Programming (AOP) [11,12]. In AOP the models of data processing are represented as a model of automata and the program unit describes the work of finite state automata. The main features of the AOP are:

1. The model of data processing is specified as a model of automata represented as a transition matrix, transition diagram or state diagram. States of the automata are defined depending on the values of subset of variables from the set of variables which is de-

fined when problem analyzing. Thus the states of automata differ one from another through different values of the variables which describe the problem. In the time of automata step execution some variables obtain new values. In modeling of composed data processes several diagram may be used. The method of diagram association is specified in particular way depending on problem peculiarities.

2. The program unit contains the loop. The body of this loop represents the program code fragment which describes a work step of the automata. This program code fragment can be represented as a subprogram – function. The program execution is driven by events produced by external or internal actors. AOP can bee realized by means of structured or object oriented programming.

The methods of transition diagram application in designing of languages processors is described in many publications [2,4,14]. The methodology of interpreter modeling by means of transition diagrams is a good base for development of methodology of CIS dynamic project construction.

The methodology of AOP originates a new programming paradigm – Event-Driven Programming (EDP) [15-17]. The general static model of event-driven application is shown in Fig 1.

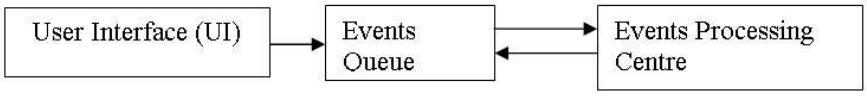

Figure 1. The general static model of event-driven application

The UI component offers to user the means of event generation: windows with menus, buttons, etc. The event generation is realized by the user through different ways: mouse clicking a menu line, a button,

etc. The ,,Events Queue" component ensures the events saving in a queue. The ,,Events Processing Centre" component takes the events from the queue and processes it. The model of events processing is specified by a transition diagram and the interpretation of this component functions is driven by this diagram. The events can be processed in parallel way [16].

Currently the EDP is developed by means of OOP [16]. The author used this technique in elaboration of the "Electronic Atlas of Moldova Republic" application prototype [18]. But EDP can be developed by means of structured programming and STD [19]. See compartment 5.

4 Deterministic STD

From theoretical point of view STD can be considered as the association of the transition diagrams realized by means of the standard method of association. Therefore the STD structure can be modified easyly.

From practical point of view it is very important that the STD should provide a deterministic transition from one to another state.

A TD of STD uses two disjoint vocabularies: vocabulary of events names and vocabulary of diagrams names. The set of events can be defined from specification of functional requirements to the application. This set is finite always. One can define the events sequence for each functional requirement. The detailed analysis of the construction method of STD is not provided in the context of this paper. Two general models of STD construction may be pointed out.

Model 1 includes the following common steps:

St1) The functional requirements are defined as sequences of events names.

St2) The transition diagrams of STD are constructed on the basis of sequences of events names having the common prefixes or/and suffixes.

Model 2 provides for the elaboration of the events set and the STD in the same time as the top-down functional designing of a CIS.

As a rule the STD constructed by applying any method is a non-deterministic one. In this case it is necessary to check STD and if it is not deterministic then transform it into deterministic one. To define

the STD which models the deterministic process, the formal notations were proposed [6]:

Σ – vocabulary of events names;

N – vocabulary of diagrams names;

d_i – the name of i-th diagram of the STD ($i \in 0...n-1$) and $d_i \in N$;

IL – input line;

Σ_{ij} – the set of events names, which mark edges going from j-th node of i-th diagram;

N_{ij} – the set of diagram names, which mark edges going from j-th node of i-th diagram;

$F(d_i)$ – the set of simple events which are the first events of the sequences that can be interpreted under passing d_i diagram ($i = 0...n-1$). The construction method of $F(d_i)$ set ($i = 0...n-1$) is:

$$F(di) = \Sigma_{i0} \cup (F(d_l)), \text{ for all } \ d_l \in N_{i0}$$

These notations helped to define exactly the limitations for diagrams structure which ensure the deterministic work of the diagrammer. The limitations for diagrams structure are:

1) $N \neq \emptyset$;

2) Σ and N vocabularies don't contain useless symbols;

3) STD diagrams don't contain unmarked edges;

4) There can't be two edges, outgoing from the same node and marked with the same symbol;

5) For every d_l diagram ($d_l \in N_{ij}$) the intersection of Σ_{ij} and $F(d_l)$ is the empty set;

6) For 2 diagrams d_i and d_l ($d_i, d_l \in N_{ij}$) the intersection $F(d_i) \cap F(d_l)$ is the empty set.

The sets $F(d_i)$ (i =0...n-1) can be calculated for concrete STD before the diagrammer starts execution.

It was proved, that if STD diagrams satisfy enumerated limitations, then in every state and current event the diagrammer can choose only one possible transition [6].

On the base of the diagrammer constructed in [6] the deterministic model of events sequence interpretation was elaborated. It was

named AIM (Application Interpretation Model). The general description of the AIM was made up by means of pseudo language based on C language. See Appendix 1.

The knowledge about limitations for diagrams structure and AIM simplify the process of STD construction.

5 Applying of the STD in the development of software systems

The CIS's dynamic project representation by means of STD correlates very well with its top-down designing. Top-down design methodology of the complex program systems recommends initially to develop a general structure of a system - its components (subsystems), and after that it recommends to do detailed modeling of every subsystem. In this case a dynamic project can be represented by a STD, which contains a main diagram named "PS". This diagram specifies the behavior of program system's subsystems. The STD also must contain at least one diagram for every component. These diagrams specify the behavior of subcomponents of the appropriate components.

Let us consider an example of a generalized project of complex software system, which consists of subsystems $c1$, $c2$, $c3$, $c4$, $c5$. Let these subsystems realize the following functionalities, necessary to user: $c1$ – data initializations and adaptations, $c2$ – calculations according to the method M1, $c3$ – calculations according to the method M2, $c4$ – comparative analysis of the obtained results, $c5$ – report generation. Suppose that $c1$, $c2$ and $c3$ components need an intensive interaction with the user. In this case, the behavior of the systems may be specified by the main diagram "PS", shown in Fig. 2.

Symbols "another problem", "repeat" and "exit" are the names of the simple events and symbols $c1$, $c2$, $c3$, $c4$, $c5$ are the names of diagrams which specify the components behavior. These names may be considered as names of composed events. To simplify understanding, the names of the program units associated with events are not shown on the diagram edges.

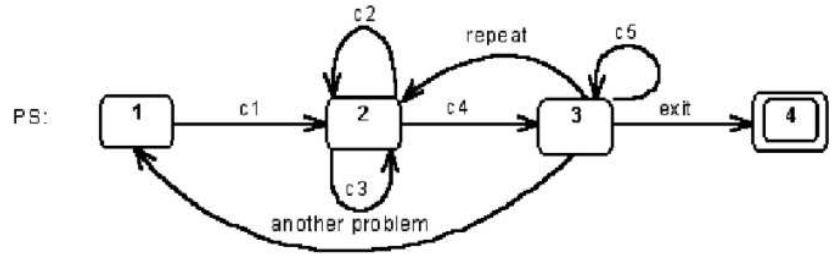

Figure 2. The main diagram of the SDT

Suppose that the diagram, which corresponds to component $c1$, has the structure, shown in Fig.3. The symbols $e0$, $e1$, $e2$, ..., eN are the names of simple events in this diagram. The $F(c1)$ set includes **e0** symbol only.

The transition diagrams for components $c2$, $c3$, $c4$, $c5$ of the CIS are constructed similarly.

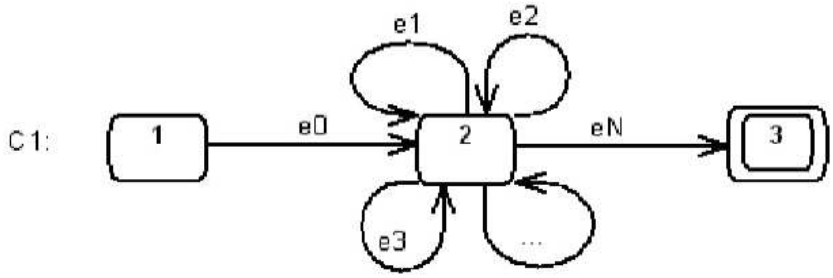

Figure 3. The transition diagram for subsystem $c1$

The correctness of one diagram structure can be tested. In this case the pointed diagram is used as a main diagram and AIM is interpreted

for selected sequence of events names. For example, we can apply the AIM by using the diagram c1 as a main diagram. For that we must fix the $c1$ as the main diagram and we mast prepare the sequences of events (user functionality) accepted by $c1$ diagram. Then it is necessary to interpret this sequence of events according to the AIM. Let the 'e0e1e1e3eN' be prepared to satisfy the user function 1 and the sequence 'e0e2e1e2e1e3eN' – the user function 2. We can make sure that the AIM accepts these sequences.

It is easy to observe that to modify the application it is necessary to modify the STD, but the AIM remains the same. It is very important to notice that the modification of the STD which isn't violating the foregoing limitations don't break the AIM work. Therefore the AIM can be implemented one time and can be reused when constructing the other application.

Practically it is important to outline the classes of problems which can be solved by application of a STD. Evidently the STD can't be used for the systems development for parallel data handling. But it can be used with success on elaborating the structured software systems for management automation.

If the dynamic project is represented by a single transition diagram and the user interaction is required not frequently then one can consider that application of the STD and AIM is less efficient.

If the IHP is very complex and his execution requires an intensive interaction with the user or other actors, then applying the STD is efficient enough. In this case it is very important the generation mode of the sequence of events. There can be used three modes or strategies of driving the AIM's work:

(1) Any functionality of a CIS is defined preliminarily by a finite sequence of the events. This sequence is placed in the queue at the beginning of AIM work. The AIM interprets automatically events from the queue.

(2) The event is placed in queue under every transition to new state of a diagram. The AIM waits the event in every state.

(3) AIM's work is partially driven by finite subsequences of the events, and partially by the event generated under transition into a

new state.

To provide the strategies (2) and (3) it needs to execute some actions under entering the state and under exiting the state. Thereby AIM functioning can be adapted in correspondence with the strategy of AIM driving.

To raise the efficiency of event-driven CIS elaboration process the model of framework was proposed [19]. The static model of the framework is shown in Fig. 4

The ,,UI constructor" component helps to construct the user interfaces of applications. The ,,Librarian" component helps to create and modify the library of program units. The ,,STD Constructor" component helps to construct the inner representation of deterministic STD. The ,,CIS integrator" component integrates together three components (AIM, UI, and STD) and creates an interpretable CIS. The "CIS interpreter" component customizes and starts running the CIS.

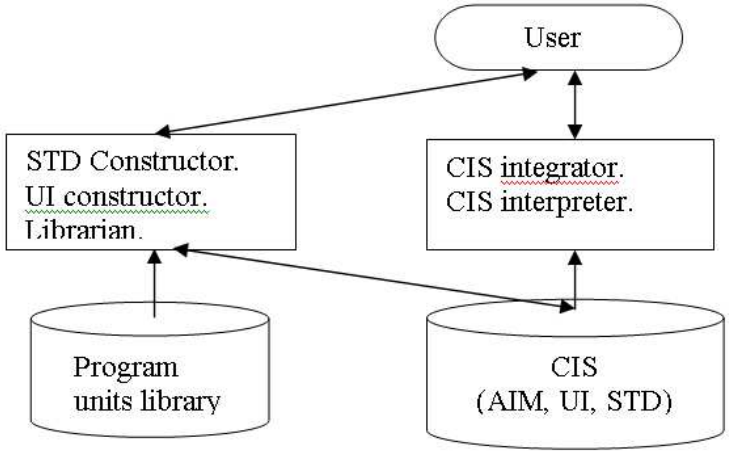

Figure 4. The static model of the framework

The analysis of the functioning principles of the framework per-

mits to affirm that the elaboration process of an event-driven CIS with applying the STD will minimize CIS elaboration time and cost.

6 Conclusions

One can mention that using the STD on software system development offers some important advantages:

1. Application of STD represents a more systematized technique of modeling of complex software system behavior than other known methods. It permits the effective distribution of work among developers and fast assembling of CIS.

2. The AIM represents an invariant part of an application. Therefore it can be constructed once and reused on developing of other CIS. This fact provides the possibility to expand the quality of software systems and reduces time of their construction.

3. The structure of a software system can be modified and extended fast and easy.

4. The automation of the STD creation can reduce the time of CIS construction.

5. STD and AIM represent an efficient mechanism for elaborating event-driven software systems.

To construct quickly the compound software system, the instrumental framework may be constructed using the results stated in this paper.

CIS design by means of STD is based on functional refinement of the project. To improve effectiveness of this type of modeling it needs to be provided with an adequate systematic method of data design and control.

References

[1] Stepan A., Petrov Gh., Iordan V. *The fundamental for design and implementation of software systems.* /MIRTON ed., Timisoara, 1995. 282 p. (In Romanian)

[2] Ciubotaru C.S., Magariu N.A. *Scanners elaboration.* (The workbook.) /State university of Moldova, Chisinu, 1984. 28p. (In Russian)

[3] http://yourdon.com/strucanalysis/wiki/index.php?title=Chapter 13, *#STATE-TRANSITION DIAGRAM NOTATION*, Retrieved January, 2008

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

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

[6] 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)

[7] Magariu N.A. *The APL programming language.* /Radio i sviazi, Moscva, 1983. 87 p. (In Russian)

[8] Magariu N. *The using of transition diagrams on systems construction of data processing.* //Proceedings of republican scientific conference "Informatics and computer engineering", Chisinau, 1993, pp. 61–62. (In Romanian)

[9] Ziubin V.E. *The programming of information and management systems based on finite automaton.* (The workbook.)/ State university of Novosibirsck. Novosibirsk, 2006. 96 p. (In Russian)

[10] David Harel. *Statecharts: A visual formalism for complex systems.* //Science of Computer Programming 8, North Holand, 1987. pp. 231–374.

[11] *The software technology based on finite automaton* / http://www.softcraft.ru/auto.shtml#ap. Retrieved January, 2008

[12] *The programming based on automaton* / http://is.ifmo.ru/. Retrieved January, 2008 (In Russian)

[13] Cuznetov B. P. *The psychology of programming based on automaton.* // BYTE/Russian, 11, 2000 and www.softcraft.ru/design/ap/ap01.shtml. Retrieved January, 2008 (In Russian)

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

[15] Shalito A. A., Tokkeli N.I. *The automaton implementation under programming of event-driven sytems.* // A Programer, 4. St. Petersburg State institute of fine mechanics and optics, 2002. pp. 74–80. (In Russian)

[16] Stephen Ferg. *Event-Driven Programming: Introduction, Tutorial, History.* Version 0.2 – 2006-02-08. / http://Tutorial_EventDrivenProgramming.sourceforge.net. Retrieved January, 2008

[17] N.N.Nepeivoda. *Programming styles and methods.* Internet University of Information Technologies intuit.ru, 2005, 320p. (In Russian)

[18] Magariu N., Grico I. *The design dimension of the electronic geographic atlas of Moldova Republic.* //Scientific annals of "A.I. Cuza" university of Iaşi. Number 6. The proceedings of GIS symposium. V. XLVI. Iaşi, 2000. pp.55–60. (In Romanian)

[19] Magariu N. *Some modern peculiarities of development of complex application.* // Microelectronics and Computer Science, int. conf. (5, 2007, Chişinau), Proceeding of the 5^{th} International Conference on ,,Microelectronics and Computer Science", sept.19-21. 2007, Chişinau, Moldova/ com. or: I. Balmuş, V. Ababii,... Ch., U.T.M, 2007. ISBN 978-9975-45-047-8 (vol2). pp. 149–152. (In Romanian)

APPENDIX 1. The Application Interpretation Model.

// AIM

//The initial work state of the AIM is fixed.

i = 0; //The main diagram was selected

j = 0; // The initial nod of diagram was selected

{The content of the input line *IL* is fixed};

k = 1; //k – the index of a symbol from the *IL*.

while (the current node is not the final node of the main diagram)

{ ***while*** **(**IL_k $F(d_l)$ and d_l N_{ij}**)**

{The current state is memorized in stack;

The initial state of diagram $d_l$is fixed as current state; // Values of $i$and $j$variables are modified.

}

if **(**IL_k Σ_{ij}**)**

{ The program unit associated with the edge, marked with IL_k is executed;

The transition through the edge, marked with IL_k is realized and the value of j variable is modified;

k = k + 1;

}

else

{ Message delivery "The sequence of control symbols is wrong"; **exit** (1)}

while **(**The current node is the final node of a diagram that differs from the main diagram**)**

{ The state from the top of the stack is determined as the current state;

The transition through the edges, marked with name of passed diagram, is made;

}

}

{The interpretation is stopped};

Nicolae Magariu

Received February 4, 2008

State University of Moldova,
Mathematics and Computer Science Department,
Research Laboratory "Software System Designing"
E–mail: *magariu@usm.md*