KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2007
Cluj-Napoca (Romania), June 6–8, 2007, pp. 346–352

APPLYING TRANSITION DIAGRAM SYSTEMS IN DEVELOPMENT OF INFORMATION SYSTEMS DYNAMIC PROJECTS

NICOLAE MAGARIU

Abstract. The model of complex software development based on the applying of transition diagrams systems is considered in this paper. This model provides the minimization of elaboration time, improving the quality, and raising the technological level of the software.

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 elaboration needs big intellectual efforts of a group of developers in the course of several years. In the context of CIS elaboration 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, 3]. 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 solution of the posed problem. In this paper a model of applying 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 elaborated a separable compiler [4]. Conway had specified

2000 *Mathematics Subject Classification.* 68N30, 68N99.
Key words and phrases. Dynamic project, Systems of Transition Diagrams, Complex Information System, diagrammer.

the syntax of COBOL language - a Symbolic Programming Language (SPL) - utilizing STD, and had proposed an original algorithm of compilation which he had named Diagrammer.

A diagram defines the syntactical structure of an SPL construction and has a name, which identifies this construction. The diagram consists of nodes and edges, where nodes represent states and edges define transitions from one state to another. Edges can bee marked by terminal symbols of the SPL or by name of a diagram. The method of interlinking between diagrams characterizes a STD. The STD includes a main diagram - "program". Every edge can be associated with a program unit, which is executed when the corresponding edge is passed. The diagrammer analyzes a program (an input line) and identifies the program construction using one of the diagrams from STD. The diagrammer starts the input line analysis from the initial state (node) of the main diagram and recognizes program constructions in nondeterministic way - with returns in input line. Conway determines the possibility of STD and diagramer application for other languages compilation.

The American specialist, David Bruce Lomet, had studied the Conway diagrammer and had proved diagrammer's equivalence to a restricted Deterministic PushDown 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 diagrammer independently of D. B. Lomet. A new version of diagrammer that functions in deterministic way was elaborated by him [6]. It was applied in the elaboration of APL interpreter.

The syntax of APL language constructions is very simple [7]. User variables are dynamic and they can obtain as a value a numerical array or an array of char type with arbitrary number of dimensions. APL operations can be nullary, unary or binary. Operands of the operations can be some expressions, evaluated to arrays, with arbitrary number of dimensions. Semantics of the APL operations is very consistent and can be described in C language using tens or hundreds of instructions. Almost all operations in APL have equal priority (with small exceptions). User Defined Functions (UDF) prototypes have the structure similar to the APL basic operators structure. The 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 class of problems are stored in a Work Space. An expression can contain invocations of UDF. To solve a problem by means of APL system, one needs to compose the set of UDF and the APL expression. The execution of this expression leads to execution of UDFs, which are invoked directly or indirectly from this expression.

So, the SDT of the APL expression defines the behavior of a UDF system. The APL expression represents a control message to realize functionality, which

is necessary for a user. The diagrammer controls the order of execution of UDF (program units). In such way, the author came to the idea about using the SDT in designing of CIS [8]. In this case the SDT represents a dynamical project and an input line represents a sequence of control symbols for the diagrammer. Diagrammer takes control symbols from input line and executes corresponding program units of the project.

3. Diagrammer functions

The STD uses two disjoint vocabularies: Σ – terminal symbols vocabulary (control symbols), N – nonterminal symbols vocabulary (diagrams names).

The author proposed the set of formal notation for STD and defined some diagram structure limitations. These limitations provide the deterministic diagrammer work. The following notations were introduced:
Σ – terminal symbols of vocabulary;
N – nonterminal symbols of vocabulary;
IL – input line;
d_i – the name of i-th diagram of the STD ($i \geq 0$), $d_i \in N$;
Σ_{ij} – the set of terminal symbols, 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 terminal symbols which are the first symbols of the sentences that can be read on passing d_i diagram ($i = 0..n-1$).

The construction method of $F(d_i)$ set:

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

The limitations for STD diagrams:

1) $N \neq \emptyset$;
2) Σ and N vocabularies don't contain inaccessible 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 Σ_{ij} and $F(d_l)$ is empty;
6) for 2 diagrams d_i and d_l, ($d_i, d_l \in N_{ij}$) the intersection $F(d_i)$ and $F(d_l)$ is empty set.

The sets $F(d_i)$ for concrete STD can be calculated before diagrammer starts execution.

If STD diagrams satisfy enumerated limitations, then in every state the diagrammer can choose only one possible transition. So, the diagrammer functions in deterministic way.

The determinate diagrammer executes the following actions:

A1. Work state of the diagrammer is determined: $i = 0; j = 0;$.

A2. The contents of the input line IL is determined; $k = 1$; $//k$ – the index of a symbol from the IL.

A3. If $IL_k \in \Sigma_{ij}$, then: {
- Program unit associated with the edge, marked with IL_k is executed;
- Transition through the edge, marked with IL_k is realized and the value of j variable is modified;
- $k = k + 1$;
- Go to $A5$;}

A4. If $IL_k \in F(d_l)$ for d_l which belongs to N_{ij}, then: {
- The current state (the number of the current diagram and the current node number of this diagram) of the diagrammer is memorized in stack;
- It is fixed as current state the state corresponding to initial one of d_l diagram and respectively modifies values of i and j variables.}

If N_{ij} doesn't contain a dl diagram for which $IL_k \in F(d_l)$, then an error is detected in input line and the diagrammer stops.

A5. If the current node is the final node of the main diagram then the diagrammer stops.

If the current node is the final node of a diagram that differs from the main diagram then: {
- The information on the top of the stack is deleted;
- The state from the top of the stack is determined as the current state;
- The transition through the edges, marked with the name of the passed diagram, is made;}

Go to $A3$; //Or ***accepts an event*** after which goes to $A3$.

4. The applying of STD and diagrammer in the development of information system's dynamic project

When a STD represents a dynamic project of a software product (SP), we can naturally assume, that input line is always correct and so it is always accepted by diagrammer.

If the dynamic project is represented by a single transition diagram and this program realizes a single functionality, which doesn't need an intense dialog with the user, then we can consider the applying of the diagrammer less efficient.

If the IHP execution requires an intensive interaction with the user or other actors, then applying the diagrammer is efficient enough. In this case the diagrammer can work in conditions of accepting messages.

Depending on the way of interaction between user and CIS, elaborated on basis of diagramer, there can be used three modes of driving the diagrammer's work:

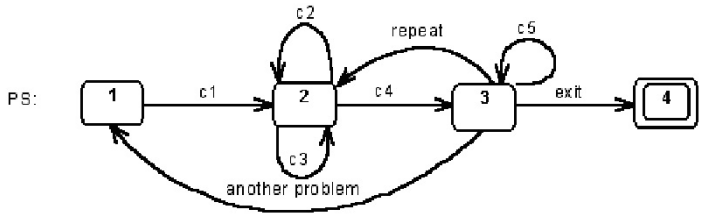

FIGURE 1. Main diagram of the SDT

(1) Any functionality of a CIS is defined preliminarily by a constant sequence of the control symbols. This constant is placed in input line of the diagrammer after selecting the corresponding option from a menu.
(2) Control symbols are generated as a result of the events produced by users or some other actors. The diagrammer waits the appearance of an event from the given set of events.
(3) Diagrammer's work is partially driven by constants - sequences of the control symbols, and partially by events.

Diagrammers functioning can be adapted in correspondence with the exploiting requirements of a CIS.

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

Let us consider an example of a generalized project of complex software system, which consists of components $c1$, $c2$, $c3$, $c4$, $c5$. Let these components realize the following functionalities, necessary to user: $c1$ – initializations, adaptations, $c2$ – calculations according to methodology M1, $c3$ – calculations according to methodology M2, $c4$ – comparative analysis of the calculation results, $c5$ – report generation. Suppose that $c1$, $c2$ and $c3$ components need an intensive interaction with the user. Suppose that the behavior of the components is specified by the main diagram "CIS", shown in Figure 1. Symbols "another problem", "repeat" and "exit" are terminal symbols (events).

Suppose that the diagram, which corresponds to $c1$ component, has structure, shown in Figure 2. In this diagram $e0$, $e1$, $e2$, ..., eN are the terminal symbols (events). The names of the program units associated with edges of the diagram are not shown on diagrams. On appearing of the event the diagrammer starts

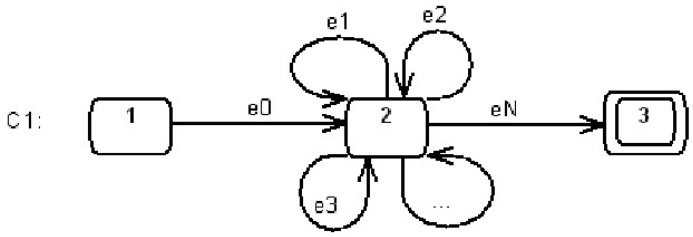

FIGURE 2. Transition diagram for component c1

the execution of the program unit, associated with the corresponding edge of this diagram. The transition is done after execution of program unit.

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

It is easy to observe that the modifying of the software system needs the STD modifying only, but the diagrammer remains the same.

5. CONCLUSIONS

We can mention that the application of STD in software system development offers some important advantages:

1. Application of STD and determinate diagrammer represents a more systematized method of modeling complex software system's behavior and construction than known analogical methods. It permits the effective distribution of work among developers and permits fast assembling of software system.
2. The diagrammer represents an invariant part of a software system. Therefore it can be constructed once and reused in 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 reduces time of CIS construction.
5. Deterministic STDs and diagrammer 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 stated results 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 completed with an adequate systematic method of data design.

6. References

1. A. Stepan , Gh. Petrov , V. Iordan , *Fundamentele proiectării şi realizării sistemelor informatice*, Ed. MIRTON, Timişoara, 1995, 282 p.

2. Ciubotaru C.S., Magariu N.A., *Organizaţyia lexicescogo analiza* (Metodicescaia razrabotca). /RIO CGU, Chişinău, 1984, 28 s. (In Russian)

3. *http://yourdon.com/strucanalysis/wiki/index.php?title=Chapter_*13 *#STATE- TRANSITION_DIAGRAM_NOTATION*

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. *Ispolizovanie diagram perehoda pri realizaţii dialogovoi sistemi programmirovania.* //Matematiceschie issledovania AN MSSR, vîpusc 107. Teoria i practica programmirovania. Chişinău, Ştiinţa, 1989, ss. 100–110.(In Russian)

7. Magariu N.A., *Iazîc prorammirovania APL.* /Radio i sviazi, Moscva, 1983, 87 s. (In Russian)

8. Magariu N., *Utilizarea diagramelor de tranziţie la construirea sistemelor de prelucrare a informaţiei*, Materialele conferinţei republicane "Informatică şi tehnică de calcul", Chişinău, 1993, pp.61–62.

State University of Moldova, Chişinău
E-mail address: magariu@usm.md