DOI 10.29042/2018-2295-2300

Helix Vol. 8(1): 2295 – 2300

Transformational Technique of Interactive Systems Development

Arslan I. Enikeev[1], Victor O. Georgiev[2], Rustam A. Burnashev*[3]

[1, 2, 3] Kazan Federal University, Institute of Computational Mathematics and Information Technologies

Email: r.burnashev@inbox.ru, **Contact**: 89274391734

Received: 21st October 2017 Accepted: 16th November 2017, Published: 31st December 2017

Abstract

In recent years there has been a great deal of interest in developing the interactive systems. A wide variety of class of prob1ems requiring an interactive way of solution and new facilities represented by computers project development of the new interactive systems. In order to increase the efficiency of a development process and to attain high qua1ity of these systems we need the appropriate technological tools. Among such technological tools transformational technique of software development seems to be very useful [1,2] . This technique provides a possibility of computer-assisted transformation of programs according to given transformation rules to generate new programs or to make reduction (optimizing) of existing ones. In this report we consider an app1ication of transformationa1 technique to interactive system development. We represent a conceptua1 model of an interactive system with the special functions providing an interactive control and methods of transformation of the "scenario" type interactive system into equivalent system with the "built-in" program interaction. Such transformation seems to be reasonable in case of repeatedly reusing the same scenario in order to increase the efficiency of a dialogue interaction. The conceptions and methods represented in the report were applied to the development of interactive tools supporting CAD , Data Base and CAI- systems [3].

Keywords: Interactive Systems Development, Principles of Transformational Technique, Dialogue Interaction.

Introduction

The Principles of Transformational Technique.

An implementation of transformational technique requires a conceptualization of the basic programming constructions. Therefore we need an appropriate formalism to provide a specification and analysis of the respective constructions. Hoare's calculus [4] seems to be well suited to the requirements.

Let the variable names x, y,... denote the values taken by these variables before execution of a statement P. Use dashed variables x', y', … to denote the possible final values which may be taken by these variables when the statement P terminates. A specification of statement P can now be formulated as a predicate P (x, y, … x', y', …) which shows how the variables x, y, … changed after terminating the statement P.

We shall assume that we are interested only in program variables x and y: the discussion is readily adapted to different sets of program variables.

Methods

Identical Statement

IDENT = df(x' = x & y′ = *y*).

Assignment

Let P be a specification, let x be a program variable and e be an expression. Let (e $--_x--->$P) be formed from P by replacing every free occurrence of x by e. Then:

(x : =e ; P) = df ¬ (De) v (e $---_x--->$ P) ,

where De specifies the domain of the expression e.

Examples:

1. x: = x + y; IDENT= (x '= x + y &y ' =y);
2. (y:= y / x ; (x:= x +y ; IDENT)) = (x=0 V x'=x+ y / x & y '= y / x);

Conditional

Let P and Q be specifications, and let b be a logical expression, containing only unlashed variables. Then:

if b then P else Q = df ¬ Db V b & P V ¬ b & Q

where Db specifies the domain of expression b.

Sequential Composition.

Let P be a specification containing the variable p, which itself ranges over possible assertions. If Q is a specification we define (Q $--_p--->>$P) as the result of replacing all occurrences of p in P by Q. Construction P may be regarded as describing the behavior of an assembly with a slot p into which a range of

different components may be plugged.

Let SKIP be a special assertion variable which by convention stands for an assertion describing observations that can be made after an appropriate statement successfully terminates. We can now define a sequential composition (P;Q) as a construction which behaves like P up to the moment that P successfully terminates; and thereafter behaves like Q :

P;Q = df (Q --- SKIP -->> P)

For the sake of brevity we omit here a formal definition of "successful termination" introducing only the examples of unsuccessful termination: (while true does P;Q), y:=x/0; Q.

While Loop.

Let b be a logical expression and P be a specification. Then :

while b do P = df if b then (P ; while b do P) else SKIP

The following assertions can be readily deduced from definitions :

1. Db ==> (if b then P else P) = P
2. if b then P else Q = if ¬ b then Q else P
3. (x:=e; if b then P else Q) =if (e ---x---> b t hen (x: = e; P) else (x:=e ; Q)
4. SKIP; P = P; SKIP = P
5. P; (Q; R) = (P; Q) ; R
6. if b then P else Q) ; R = if b then (P;Q) e1se (Q;P)
7. if n is a variable occurring only in Pn , then :
   - (x: = e; ∀n.Pn) = (∀n . x: = e; Pn);
   - if b then (∀n.Pn) else Q = (∀n. if b then Pn else Q);
8. if b then (while ¬ b do P) = SKIP

Such algebraic identities represent transformation rules to support the program transformations.

Addressing to logic programming provides the rich facilities to support transformational technique, because in addition to transformation rules related to the programming feature it is possible to apply the rules of mathematical logic. Note, that an application of transformational technique to the development of interactive programs requires formalizing communication tools to specify a man - machine interface. A calculus which seems to be very useful for that aim is represented by Hoare's theory of communicating processes [5,6].

According to this calculus a process communication is provided by synchronized output and input on named channels. A command c?x denotes outputs a message m along the channel c. A command inputs message on the channel c and assigns this value to the variable x. Note that the event of outputting a value m along a channel c is exactly the same event as inputting a value m along the same channel.

This means that output in one process can occur on channel c only at the same time as input of the same value along channel c in the other communicating process. The communication can actually take place only when environment of mechanism is ready to communicate on that channel. To indicate the readiness we use a special variable c which takes the value "true" when the mechanism is ready to communicate on channel c and takes the value "false" otherwise. On each named channel, it is possible to keep a record of all messages passing along it. At any given moment, the record of all messages that have passed so far on a channel c is a finite sequence which will be denoted c = <m1, m2, … ,mn>, where every mi (i = 1,n) is the message passed along channel c. At the very beginning the value of c is the empty sequence <>. To specify communication commands we will use two channels c and d .

Empty Process.

E = df (c= <>) & (d=<>)

Actually this process describes an initial state

of communicating process with the channels c and d.

Output.

c ! e; P df = (¬ De) V (E & c) V

(c>= <e>) & (c := tail (c) ; P),

where De specifies a domain of the expression e ,

tail (c) is a sequence resulted from sequence c after removing the first element of c.

Input.

d?x; P = df (E & d) V

(d ><>) & (x:= head (d) ; d: =tai1(d); P),

where head (d) is the first element of sequence d.

Alternation.

Let P and Q be specifications which begin with input o r output commands. Then:

P □ Q =df (E & P & Q) V ¬ (E) & (P V Q)

This operation specifies a process that is initially ready to engage in either of the two initial communications : after executing one of them, readiness to execute the other is withdrawn, and execution continues with the process whose initial command was selected. (P□Q) differs from P V Q only in its initial state, in which its readiness to communicate is the conjunction of the readiness of P and of Q.

Results and Discussion

A Model of the Scenario Type Dialogue Interaction.

A general flowchart describing an interactive system behavior includes the following actions :

1. message (prompting) output;
2. user's message input;
3. running modules (functions or procedures) selected in accordance with the given scenario dependent on user's input;
4. selection of the next step of dialogue interaction which is also determined by scenario and user's input.

Therefore every step of a dialogue interaction can be specified as a lists s = (M , P , N), where M is an output message, P= (p1 , p2,…, pk) is an alternative set of functions (procedures) and N = (n1 , n2 , … , nm) is a set of integers denoting one of the possible next steps of dialogue. Selection of pi (i = 1, k) and ni (i = 1 , m) depends on the value of a variable (say x) representing a user's message.

Assume that a selection of ni=0 provides terminating a dialogue interaction (all other ni>0, i = 1,m). Callthe set s scenario element and the list S= (s1, s2,…,sn), where every si (i=l,n) is a scenario element, call scenario of a dialogue interaction.

Let inter (S, k, x) be a specification describing behavior of a dialogue interaction with the scenario S, where k is the number of initial state of a dialogue interaction and x is a variable to input user's message. Then we can define:

inter (S, k, x) = df if $k \neq 0$ then (c ! message (S ,k) ; inter1 (S,k, x))

inter1 (S, k, x) = df d ? x; inter2 (S, k, x)

inter2 (S, k, x) = df run (S, k, x); inter (S, next (S, k, x), x),

where the function message (S, k) specifies the message output by the command " c! message (S, k)" on k step of dialogue interaction with scenario S (M); d ? x denotes input of user's

message into variable x; run (S, k, x) describes a function of selecting (running) an appropriate procedure from P dependent on the values of variables S,k,x; the function next (S, k , x) determines a number of the next step of a dialogue interaction after the k step dependent on the user's message.

We do not clarify semantics of the above – mentioned constructions because they can be implemented in different ways for the different interactive systems.

In several cases of a dialogue interaction there are possible the situations when it is required to take actions not described by the respective scenario. That is possible when a solution of problem cannot be completely formalized before execution on the computer. Therefore we need introducing the special functions to provide an interactive control. We will consider the following functions:

1. "stop" – to terminate or quit a process;
2. " back" - to restore the most previous state of dialogue interactions;
3. switch functions (" on" ," off ") - to switch from one process to another.

Function " STOP".

inters (S, k, x) = df if k ≠ 0 then (c ! (message (S, k) + " stop") ;

inters1 (S, k, x))

inters1 (S, k, x) = df (d?x; inters2 (S, k, x)) □ (stop?any ; SKIP)

inters2 (S, k, x) = df run (S, k, x); inters (S, next (S, k, x), x)

The process described by the specification " inters" behaves like "inter", except that:

1. "stop" is in every message output by interactive process;
2. when "stop" occurs, the process terminates successfully.

Note, that the variable "any" allows the values to be disregarded and a process to input if any signal is ready to be transmitted.

Function " BACK".

interb (S, k, x) = df if k ≠ 0 then (c ! (message (S, k) + " back") ;

interb1 (S, k, x) ;interb (S, k, x))

interb1 (S, k, x) = df (d?x; interb2 (S, k, x)) □ (back?any ; SKIP)

interb2 (S, k, x) = df run (S, k, x); interb (S, next (S, k, x), x)

Similar to "stop", but unlike the process "inters" , occurrence of "back" provides restoring the most previous state o f a dialogue interaction.

Function "OFF".

interoff (S1,k1,S2,k2,x) = df if κ1 ≠ 0 then

(c!(message(S1,kl) + "off"); interoff1 (S1,k1,S2,k2, x))

interoff1 (S1,k1,S2,k2,x) = df (d?x; interoff2 (S1,k1,S2,k2,x))□

(off?any; interon (S2,k2,S1,k1,x))

interoff2 (S1,k1,S2,k2,x) = df run (S1,k1,x);

interoff (S1, next(S1,k,x),S2,k2,x)

(off?any; interon (S2,k2,S1,k1,x))

Similar to previous functions except that occurrence of "off" provides switching to the process of dialogue interaction described by scenario S2.

Function "ON".

Specification of process "interon" can be obtained from "interoff" after replacing "off" by "on" and "on" by "off".

This function provides restoring action of the most previous process, interrupted by "off – function" after occurrence of "on".

Summary

An Application of Transformational Technique to the Interactive System Development.

Unlike the systems with a "built – in" program dialogue interaction, the interactive systems of scenario type provide more efficient and flexible tools for generating different dialogue interactions without special redefinition of programs. In order to generate new dialogue interaction it is only required to specify new scenario. But not in all cases we can avoid a modification of the respective programs. For example, supplementing an interactive system by the functions providing an interactive control (see chapter 2).

It is obvious that a "manual" modification (redefinition) of the programs appears to be cumbersome. Therefore we need using special programming tools to provide an efficient way of program modification. One of the reasonable solutions is an application of transformational technique to arrange an automatic modification of programs with respect to given set of transformation rules. Note, that a selection of an appropriate set of transformation rules can be made on specification level. That means a possibility of automatic transformation of one program to another when transforming the respective specification.

For example, in order to obtain a definition of process " inters" we can apply the following substitution to the definition of process "inter" :

c !message (S, k) == > c ! (message (S, k) + "stop") (to definition of "inter")

inter1 (S, k, x) ==> inter1 (S, k, x) □ (stop ? any; SKIP)

(to definition of "inter1") ,

and then rename the result of substitution in the following way:

Rename (inter == > inters, inter1 ==> inters1 , inter2 ==> inters2)

The other important application of transformational technique is a program optimization. Concerning an interactive system, that means an application of transformational technique to constructing a system with the " built-in" program interaction Ω s from interactive system (Ω, S), where Ω is a set of programs and S - scenario of dialogue interaction. Ωs represents a set of programs with the built-in program scenario of interaction - S. Such transformation seems to be reasonable in case of repeatedly reusing the same scenario in order to increase the efficiency of a dialogue interaction. For example, define the constructions "message", "run" and "next" in the following way:

message (S, k) = df element (element (S, k), 1)

run (S, k, x) = df element (element (element (S, k), 2), x)

next (S, k, x) = df element (element (element (S, k), 3), x)

where x:1.. max integer, element (S, i) = df if i=1 then head(S) else element (tail (S), i-1).

The function "element (S, k)" defines the k element of the list S.

For example, if S = ((M,(p1, p2),(n1, n2)), then message (S,1) =M, run(S,1,2) = p2, next (S, 1,2) =n2.

Restrict the scenario S to following conditions:

1) S = ((M, P, N)) (scenario consists only of one element)
2) k = 1
3) P=(p1,p2), where p1, p2 specifications of the respective procedures
4) N = (0, 0)
5) x:1..2

Note, that:
x:1..2 ==> element (P, x) = if x=1 then p1 else p2, and
run (S, 1, x) = element (P,x)

Thus we result in:
message (S , 1) = M, run (S, 1 , x) = if x=1 then p1 else p2, and
next (S, 1, x) = 0

Applying the above- mentioned equation and the respective transformation rules to the definition of "inter" we can obtains:

inter (S, 1, x) = c ! M; inter1 (S, 1, x)
inter1 (S, 1, x) = d ?x; inter2 (S, 1, x)
inter2 (S, 1, x) = if x = 1 then p1 else p2

After removing useless arguments we come to the following result:

inter (x) = c!M; inter1(x)
inter1 (x) = d ?x; inter2(x)
inter2 (x) = if x=1 then p1 else p2

It is easy to see, that as the result we obtained the definition of simple menu – select interaction process which runs the procedure with specification p1 after selection of x=1, and runs a procedure with specification p2, if x=2.

Conclusions

In several cases, it is convenient to specify programs by a set of equations representing the definite properties. A derivation of an algorithm (synthesis) from a formal specification can be made on the base of so-called rewriting rules, or more general of the conception of algebraic approach of programming. This conception, closely related to transformational technique seems to be useful for the development of interactive systems. In further publications authors hope to represent the respective results.

Acknowledgement

The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

References

[1] Ershov A.P. Mixed computation: potential applications and research issues. Proceedings of the International Symposium "Theory and Practice of Computer Software," Part 1, Novosibirsk, 1981
[2] Parasyuk I.N., Ershov S.V. The transformational approach to the development of software architectures based on fuzzy graph models. Cyberneticsand Systems Analysis, 2008, number 5, page 139 - 150.
[3] Georgiev V.O., Yenikeev A.I. Transformation Approach in Dialogue Systems Engineering. SOFTWARE & SYSTEMS. № 1, 1992, 9 – 17.
[4] C.A.R. Hoare. Specifications, Programs and Implementations. //Oxford University Computing Laboratory, PRG, 1982.
[5] C.A.R. Hoare. A calculus of total correctness for communicating processes. //Oxford University Computing Laboratory, PRG, 1981
[6] Enikeev A.I. C.A.R. Hoare. Theoretical model interacting successive processes for the dialog systems menu. Mathematics Magazine, number 3, 1987, pp 28-36

