Comput. Educ. Vol. 12, No. 4, pp. 507–512, 1988
Printed in Great Britain. All rights reserved

0360-1315/88 $3.00 + 0.00
Copyright © 1988 Pergamon Press plc

AUTOMATED CONSTRUCTION OF INTERACTIVE LEARNING PROGRAMS IN MODULA-2

KLARA VANCSO-POLACSEK and ANDREAS FISCHLIN
Swiss Federal Institute of Technology Zürich (ETHZ), Project-Centre IDA, ETH-Zentrum, CH-8092 Zürich, Switzerland

(*Received* 6 *October* 1987; *received for publication* 23 *February* 1988)

Abstract—A frame program generator written in Modula-2 is presented, which allows an automatic generation of dialogs of interactive learning programs based on the "series-parallel-repetition" user model. The formal description of a dialog program serves as input, and a Modula-2 source program is produced. The generated program is built up of empty dynamic pages which are connected with each other according to the structure defined in the input. The developing process of a learning program is illustrated by an example.

INTRODUCTION

A new methodology for the development of interactive learning programs which employs the separation of the formal aspects of dialog-controlled programs from the application specific aspects was described in [1] and [2]. Using this methodology dialog-directed programs can be produced more efficiently. The benefits are threefold:

- This approach makes an automatic construction of the dialog possible which reduces the programming effort.
- Freeing the author from the programming of the man–machine interface, the programs will be shielded from errors frequently found in dialog design.
- Different programs behave similarly, i.e. once the user learned to control one program, he is familiar with the usage of all programs constructed in this way.

Three different user models of interactive learning programs and their realization aspects are discussed in [3]. One of these user models is the "series-parallel-repetition (s-p-r)" model which has been defined elsewhere [1, 2, 4]. The detailed structure of the "s-p-r" model will be illustrated by the included sample program.

Construction of interactive learning programs based on the user model "series-parallel-repetition" can be supported by using a so called frame program generator. A portable frame program generator has been written with the UCSD-Apple Pascal-system for the Apple II personal computer [1, 2, 4]. This software transforms formally described networks into executable Pascal frame programs. Advantages of this approach are that the programming language Pascal is widely available on almost any computer and many programmers are familiar with this language. However, the main disadvantage is that inherent properties of the language Pascal do not properly support modularization. For instance the splitting of a learning program into a formal dialog controlling and subject specific part can not be reflected in the structure of the final program. Pascal forces the programmer to remerge the two parts in one single main program. Hence, teachers with minimal programming skills are confronted with large, automatically generated program sections, full of the low-level dialog controlling code, which they don't understand and which they must never touch. This violates the principle of information-hiding. Moreover, the programmer is forced to find the correct location into which he has to write his subject specific code, locations which are typically obscurely embedded in the automatically generated and hardly understood code.

Modula-2 is a programming language particular designed to support modularization, which also supports elegantly information hiding [5]. In this paper we describe our approach to use this language to solve some of the problems encountered with previous versions of frame program generators and present in details the resulting new frame program generator [6] together with an example.

THE FRAME PROGRAM GENERATOR

The here described frame program generator (FPG) generates Modula-2 frame programs based on the user model "series-parallel-repetition". The FPG Version 1. is a portable program which can be used on computers with traditional, cursor-addressable, alphanumeric CRT-displays (e.g. IBM PC). A dynamic page corresponds to a whole screen and the user controls the dialog with command keys. The FPG Version 2. was written on the Macintosh^{TM} [7] by means of the software package "Dialog Machine" [8]. A dynamic page is a window and the dialog is driven by pull-down menus, activated by the Macintosh one-button mouse.

The FPG takes as input the description of a network and produces a Modula-2 frame program containing empty procedures according to the individual dynamic pages defined in the input file (for each dynamic page a procedure with an empty body is generated). An executable dialog program is obtained by compiling (and linking) the generated program. It runs the control dialog which allows the user to move among the dynamic pages, which, however, remain empty, i.e. beside their title they display no information. Preceeding the final compilation of the complete program, the "content" of the dynamic pages has to be programmed.

The network must be expressed in a so-called LL(1)[9] formal language. A parsing algorithm can be constructed for this formal language which determines the syntax tree by scanning the input file from Left to right while Looking only **one** symbol ahead [10]. The syntax is given below using the meta language Extended Backus Naur-Formalism (EBNF):

```
network definition = >"NETWORK""("identifier")"network"ENDNETWORK".
           network = >dynpage|sequence|selection.
           dynpage = >"DYNPAGE""("dynpagedef")".
        dynpagedef = >string","identifier.
          sequence = >"SQBEGIN"network{network}"SQEND""("dynpagedef")".
         selection = >"SLBEGIN""("dynpagedef")"network network {network}
                      "SLEND""("dynpagedef")".
            string = >"""{character}"""|"""{character}"".
        identifier = >letter {letter|digit}.
```

The frame program generator consists of the following three modules:

(1) MODULE Network
(2) DEFINITION MODULE Runnet
(3) IMPLEMENTATION MODULE Runnet

The way how the frame program generator operates and a learning program is constructed is shown in Fig. 1. The program "Network" takes the network definition file edited previously by

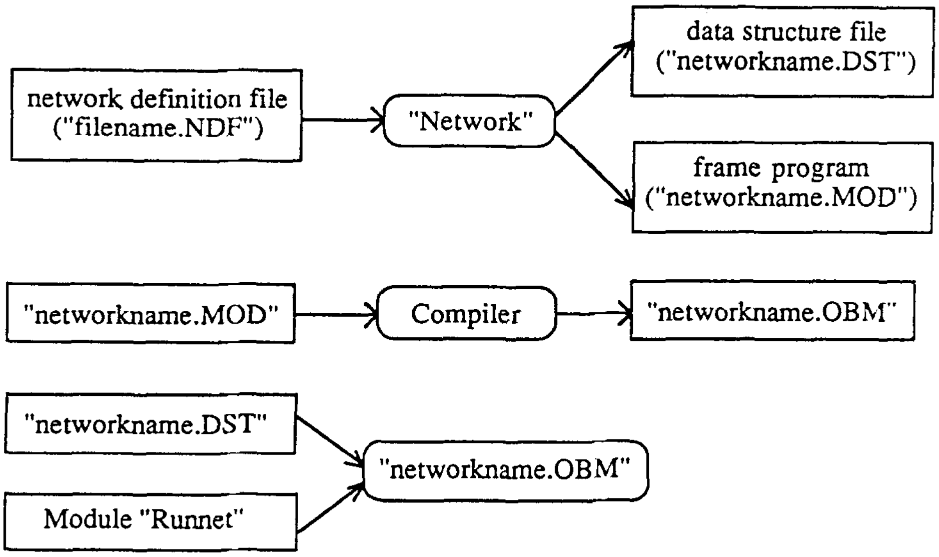
network definition file ("filename.NDF")
"Network"
data structure file ("networkname.DST")
frame program ("networkname.MOD")
"networkname.MOD"
Compiler
"networkname.OBM"
"networkname.DST"
Module "Runnet"
"networkname.OBM"

Fig. 1. Operation of the frame program generator.

```
NETWORK(Opti)
 SQBEGIN
  DYNPAGE("Help",Help)
  DYNPAGE("Set parameter",Set)
  SLBEGIN("Method Selection",Select)
   DYNPAGE("Show optimization with method 1",Opti1)
   DYNPAGE("Show optimization with method 2",Opti2)
   SLEND("Discussion",EndSelect)
   SQEND("Final remarks",EndProgram)
 ENDNETWORK
```

Fig. 2. An example network definition file. The string parameter of the individual dynamic page definition becomes the heading of the dynamic page represented in a window, and the identifier in the same parenthesis determines the procedure name in the generated program.

the user and proves its syntactic correctness. If the network definition is syntactically incorrect, error messages will be displayed with the lines containing the error. If the network definition is syntactically correct, the program Network produces a frame program written in Modula-2 for the user and an internal data structure file for the program "Runnet". The frame program contains empty procedures according to the individual dynamic pages defined in the network definition file.

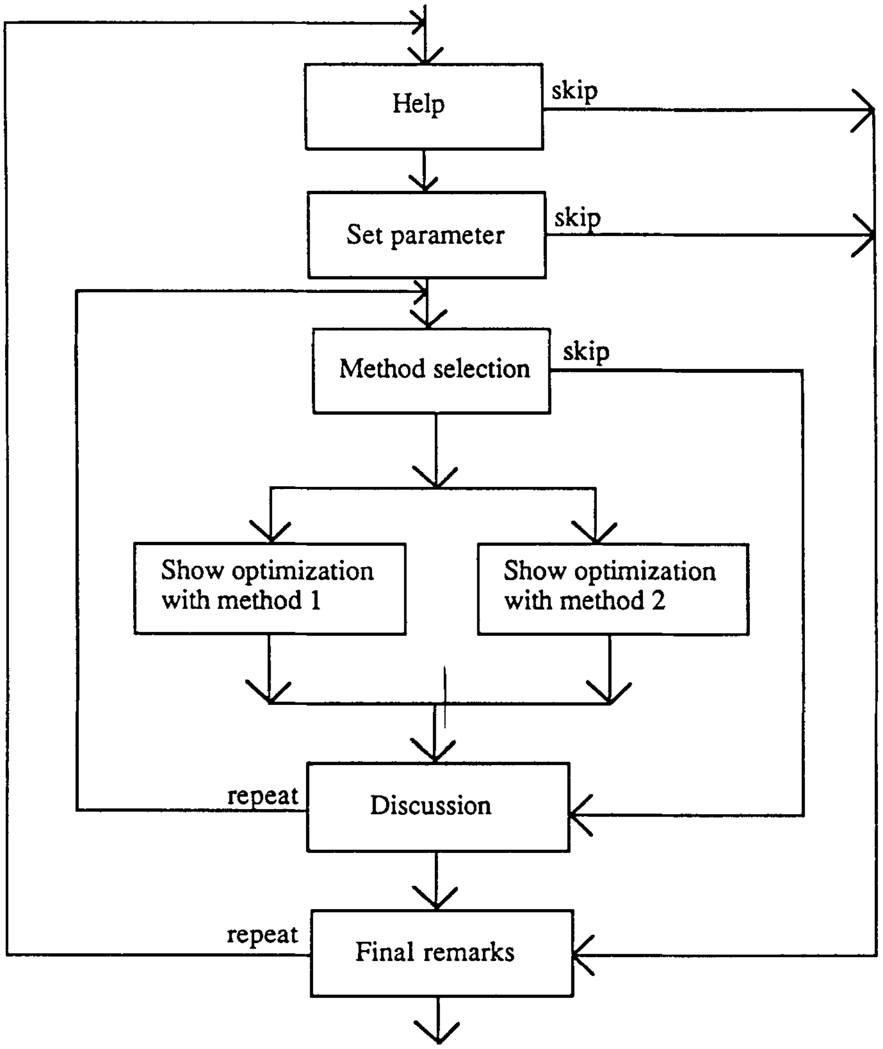

Fig. 3. The structure of the example program which is defined in the network definition file shown in Fig. 2.

```
MODULE Opti;
  FROM Runnet IMPORT Run;

  PROCEDURE Help;
  BEGIN
  END Help;

  PROCEDURE Set;
  BEGIN
  END Set;

  PROCEDURE Select;
  BEGIN
  END Select;

  PROCEDURE Opti1;
  BEGIN
  END Opti1;

  PROCEDURE Opti2;
  BEGIN
  END Opti2;

  PROCEDURE EndSelect;
  BEGIN
  END EndSelect;

  PROCEDURE EndProgram
  BEGIN
  END EndProgram;

  VAR Pages: ARRAY [0..6] OF PROC;

  BEGIN
    Pages[0]: = Help;
    Pages[1]: = Set;
    Pages[2]: = Select;
    Pages[3]: = Opti1;
    Pages[4]: = Opti2;
    Pages[5]: = EndSelect;
    Pages[6]: = EndProgram;
    Run(Pages,"Opti.DST",TRUE);
  END Opti.
```

Fig. 4. The frame program generated by the frame program generator according to the network definition file given in Fig. 2.

In the program body the names of these procedures will be passed into the program "Runnet". At this stage it is already possible to compile and execute the automatically generated program module. This is helpful for the testing of the general program behavior and allows for corrections of the overall design of the learning program at an early time (in particular before much has been invested in the programming of dynamic pages fitting poorly the general purpose of the learning program). The user has to "fill in" the bodies of the empty procedures to obtain the fully functional learning program. The executable program imports the program "Runnet" and inputs the internal data structure file generated by the program "Network".

An example network definition file is presented in Fig. 2. The corresponding program structure

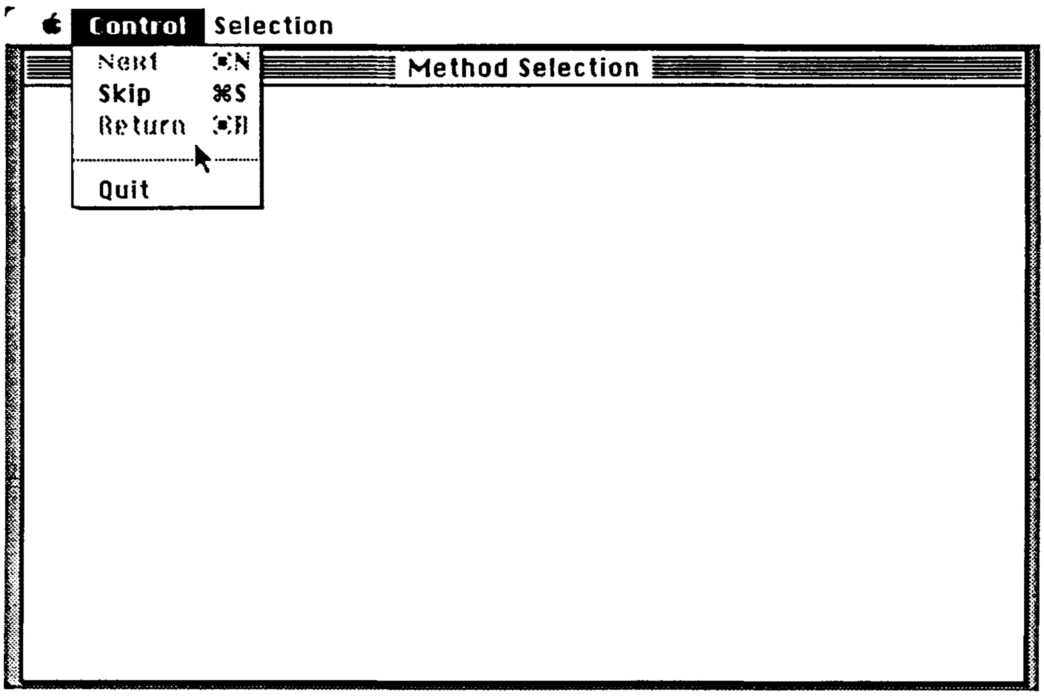

Fig. 5. A screendump of the generated and compiled program "Opti" running on the Macintosh. The actual dynamic pages is the page "Method Selection".

is illustrated in Fig. 3. The program generated by the frame program generator according to this definition can be seen in Fig. 4. Figure 5 shows a screendump of the generated program after compilation.

FINAL REMARKS

The structure of the learning program is such that the programmer is no longer confronted with the formal dialog controlling program parts which may be perfectly hidden in the module "Runnet". He can concentrate on the writing of the empty procedure bodies (Fig. 4) and is allowed to write a well structured program containing only the subject specific parts. No longer have global objects typical for learning programs to be mixed with other objects needed by the formal dialog controlling program parts. The principle to reflect logical structures in the software design can be naturally realized as well as the principle of information hiding.

It would have been possible to follow a slightly other design in order to avoid the installation of the dynamic page procedures in the frame program: namely to generate a module exporting the dynamic page procedures and containing in its implementation solely the bodies of the dynamic page procedures. However, in order to avoid recompilation of the frame part this approach requires the production of an additional program module importing the dynamic page procedures. The presented approach has been preferred over this approach due to its simplicity (one module only), although this has the disadvantage that the installation of the dynamic page procedures cannot be hidden from the programmer (see module body in Fig. 4).

REFERENCES

1. Nievergelt J. and Ventura A., *Die Gestaltung Interaktiver Programme*. Teubner, Stuttgart (1983).
2. Nievergelt J., Ventura A. and Hinterberger H., *Interactive Computer Programs for Education*. Addison-Wesley, Reading, MA (1986).
3. Vancso K. and Fischlin A., *User models of interactive learning programs*. Internal report, Project-Centre IDA/CELTIA, Swiss Federal Institute of Technology Zürich (ETHZ), 17 pp. (1987) (To be published).
4. Ventura A., *Einsatz und Programmierung des Computers als Werkzeug für den Unterricht*. Zürich, Diss. ETH Nr. 7752 (1985).
5. Wirth N., *Programming in Modula-2*, 3rd corrected edition. Springer, Berlin (1985).

6. Vancso K., *A portable frame program generator in Modula-2 for automatic construction of interactive learning programs based on the "series-parallel-repetition" user model.* User's Guide, Project-Centre IDA/CELTIA, Swiss Federal Institute of Technology Zürich (ETHZ) (1986).
7. Chernikoff S., *Macintosh Revealed.* Hayden, London (1985).
8. Fischlin A., *Simplifying the usage and the programming of modern working stations with Modula-2: the "Dialog Machine".* Internal report, Project-Centre IDA/CELTIA, Swiss Federal Institute of Technology Zürich (ETHZ), 13pp. (1986) (To be published).
9. Rayward-Smith V. J., *A First Course in Formal Language Theory.* Blackwells, Oxford (1983).
10. Wirth N., *Compilerbau,* 4. corrected edition. Teubner, Stuttgart (1986).