Dynamic Selection Screen Generation for SAP Solutions

Cornelia MUNTEAN
Faculty of Economics and Business Administration, West University of Timisoara, Romania
cornelia.muntean@e-uvt.ro

This paper presents an application for generating a dynamic selection screen in ABAP and the need and advantages of this solution instead of the manual modification of the coding itself for adding one or more parameters in the whole logic of the program. Depending on data found in a customizing table, we would like to generate a program with a selection screen, containing select options based on the criteria specified in the customizing table. The presented solution can be adapted to any dynamically generated ABAP program, according to the desired selection screen elements, and may be used as a template even for other programs. The advantages of the proposed solution are also presented along with the disadvantages, underlining the particularities of the ABAP programming language.
Keywords: *Dynamic Programming, Selection Screen, Customizing Table, ABAP, SAP*

1 Introduction

The study of algorithms forms a cornerstone of computer science research and education. However, many developers may not be aware of common algorithmic techniques, which the programming language they are using may provide. That is why companies like Google, Apple or Facebook use dynamic programming algorithms questions in their interviews and select only the best.

In many areas of activity, dynamic programming is a method for solving a certain problem by breaking it into sets of subproblems [1]. Every time the same subproblem occurs, instead of searching again for a solution, one simply looks at the previously computed solution. By using this method, precious time can be saved at the expense of a modest storage increase.

The core idea of dynamic programming is to avoid repeated work by remembering partial results and this concept finds its application in a lot of real life situations. Dynamic programming algorithms are often used for optimization purposes. These analyze if a problem has two key attributes: optimal substructure and overlapping of problems. Dynamic programming is mainly used when solutions of same subproblems are needed again and again. Afterwards, one of the following two approaches are used: a top-down approach (memorization) or a bottom-up approach (tabulation).

Since dynamic programming can be applied in so many different ways, the underlying concepts are quite general. The best way to really understand something is by making use of it, so in the next section we will check out an interesting problem which can efficiently be solved with dynamic programming, using the ABAP programming language.

2 Dynamic programming in ABAP

2.1 General Assumptions

As we know, ABAP programs may contain both static and dynamic parts, that cannot be clearly delimited. According to Cecchini [2], dynamic and static apply to the data and/or operations performed on the data within a program. It would not benefit anyone, if we would try to create categories in this regard. However, we would like to define these characteristics.

The static parts of a program refer to the characteristics of the data, which is fixed at compile time and will not change during runtime. On the other hand, dynamic properties are for various reasons not known, nor fixed at compile time [2]. Dynamic programming enables you to use certain features during runtime that cannot be determined at compile time. Instead, these features are defined by the values of

DOI: 10.12948/issn14531305/21.3.2017.05

parameters that are passed to the program at runtime.

To help us understand better the concept of dynamic programming in ABAP, we would like to present an easy example of Open SQL syntax: SELECT COUNT(*) FROM (tablename) INTO count. The variable *tablename* can be read at runtime, as the user interacts with the program by entering a value for the *tablename*. As a result, the program will be able to display a correct result for the variable, which got its value at runtime.

In a similar manner, ABAP generic data types like ANY, or DATA are used to create data references via the command CREATE DATA, which allows to dynamically specify the type of the variable at runtime [3]. Field symbols, another dynamic programming feature of ABAP, which is not found in other programming languages, help the developers access and modify data from internal tables or structures at runtime, without copying them, acting as placeholders.

We can conclude, that dynamic programming involves the use of incompletely typed or untyped data objects. This enables generic subroutines and methods to be created that run independently of the types of input data. This would not be possible in ABAP without the RTTS (Runtime Type Services), which makes it possible to handle the properties of types in programs: RTTI (Runtime Type Information) is used in order to get the corresponding types at runtime and a related service, RTTC (Runtime Type Creation), which creates new types when programs are executed and generates corresponding data objects [4].

However, dynamic programming is not only used in ABAP to define generic data objects. It can also be used to define screens and to create and run other ABAP programs at runtime. In the following section, we will explain how a complete selection screen can be generated in ABAP at runtime, without using the screen painter.

2.2 Dynamic selection screen generation in ABAP

Selection screens, unlike dynpros, can be defined through coding lines and not only by using the screen painter to define parameters, select options, radio buttons, check boxes or tabstrips. This allows the developer to hardcode his selection screen. However, if another customer requires an additional parameter in his selection screen, additional coding lines will be needed in order to satisfy the customer needs. This is a downside when you want to develop a program, which has to be used by many customers simultaneously, as it may be time-consuming to add customer specific lines of code into each transport request for every customer. The widely used solution in many ABAP programs in order to avoid this issue is to create customizing tables. Therefore, a program is written dynamically only once. It reads the customer specific data from the customizing table, and then runs according to the settings defined in the customizing table. This will also be a suitable solution in order to generate a selection screen dynamically.

As stated by Mithun Kumar [5], the existing literature, including the huge knowledge base of the internet, lacks in presenting a simple solution for achieving a dynamic selection screen generation in ABAP, although it might seem to be not a difficult issue to solve! However many developers, most of them new learners, have big difficulties to get this problem right in their codes. In this paper I want to present my application for solving this very important and useful task, although you may probably find in the near future also other solutions on the internet, but hardly one more simple and efficient.

So, in the next paragraphs I want to describe the solution I developed.

As a first step, the selection of the customizing data is in order.

The SELECT statement has been embedded in a TRY-CATCH block in order to avoid situations, where the customer does not have any data maintained in the customizing table.

DOI: 10.12948/issn14531305/21.3.2017.05

```
* Collect all information from the customizing table
  TRY.
    SELECT * FROM zparam
      INTO CORRESPONDING FIELDS OF TABLE lt_param
      ORDER BY ordr.
  CATCH cx_sy_sql_error.
    MESSAGE e460(zadapt_mess) RAISING selection_failed.
  ENDTRY.
```

This also prevents short dumps due to the missing database table, in case the customer forgets to transport the data from the test to the production system.

After selecting the data from the customizing table, the program, which has to contain customer specific coding lines, has to be generated at runtime with the parameters/select options found in the customizing table. This means, a whole new program has to be generated at runtime. As an ABAP program is an array of coding lines (each component being a character field of length 128), we will construct our program at runtime, according to the settings found in the customizing table. The generated program can be compiled, and if no errors occur, it should be pushed in the ABAP repository, activated and run as well. The compilation of the generated program can be executed with the SYNTAX-CHECK command. If errors occur, the WORD and LINE additions will tell the developer where these errors can be found; otherwise, the return code will be 0 (successful). Running the generated program occurs via the SUBMIT… AND RETURN statement. At this point, the program will call the generated program and execute it line by line; afterwards, it will return to the main program, as the statement implies.

The coding block for defining a select option (similar to a parameter, but more flexible, because it offers intervals/ranges) will contain the following lines:

```
* Selection screen with parameters from
  SELECTION-SCREEN BEGIN OF LINE.
    SELECTION-SCREEN COMMENT 1(37) s1 FOR FIELD s_1 .
    DATA so_1(80) TYPE c.
    SELECT-OPTIONS s_1 FOR (so_1) .
  SELECTION-SCREEN END OF LINE.
```

In the example above, there has been used the command BEGIN/END OF LINE in order to have only one select option per line. If there have to be more parameters or select options in one line, these can be added within the same BEGIN/END OF LINE. A selection screen comment has been used in order to add the description for the parameter/select option. The variables s1 and s_1 stand for the customizing settings, which will get their values during the INITIALIZATION event from the customizing table. Instead of the SELECT-OPTIONS command, PARAMETERS could also have been used; however, we have chosen the option above, as a select option offers more flexibility (ranges for example) in comparison to a parameter.

This coding block will have to be adapted for as many select options our customizing table may contain. Therefore, we will concatenate the strings above in a loop, incrementing the select option number in its name, and add the lines of coding to our dynamically generated program, which will be an array of character lines.

The initialization event of the called program is the step, where the each select option and its corresponding comment get their values from the customizing table. Therefore, each select option has to get values similar to the following:

DOI: 10.12948/issn14531305/21.3.2017.05

```
INITIALIZATION.

  sy-title = 'Dynamic selection screen'.

  s1 = 'Material' .
  so_1 = 'MARA-MATNR' .
```

The structure component sy-title is the system variable designated for the program title. As the select option name (so_1) gets to be concatenated in the form of tablename-component, the select option label uses the value returned by the function `'DDIF_FIELDINFO_GET'` . This function returns the value of the dictionary short text of the component in the logon language of the user.
Beside the TOP-Include, where all the data declarations occur, the INITIALIZATION block is inserted in another Include program as well with the help of the INSERT REPORT … FROM … PROGRAM TYPE 'I' statement. This is necessary for the program to be well structured and easier to read. It is important to have a COMMIT WORK statement after the reports have been inserted into the repository, as the proceeding statement is only triggered by it.

Also, the generated program has to contain comments, stating that it is created at runtime, and any changes made have no effect; instead, changed have to be made to the main program, which generated the called program. This is necessary in order to avoid confusion and wasting of precious time among developers.
The following section shows an implementation example developed by the author of this paper about how a dynamic selection screen can be generated with the help of a dynamically generated ABAP program.

3 Proposed solution for dynamic selection screen generation

We have created a customizing table called ZPARAM, which contains the fields shown in Figure 1:

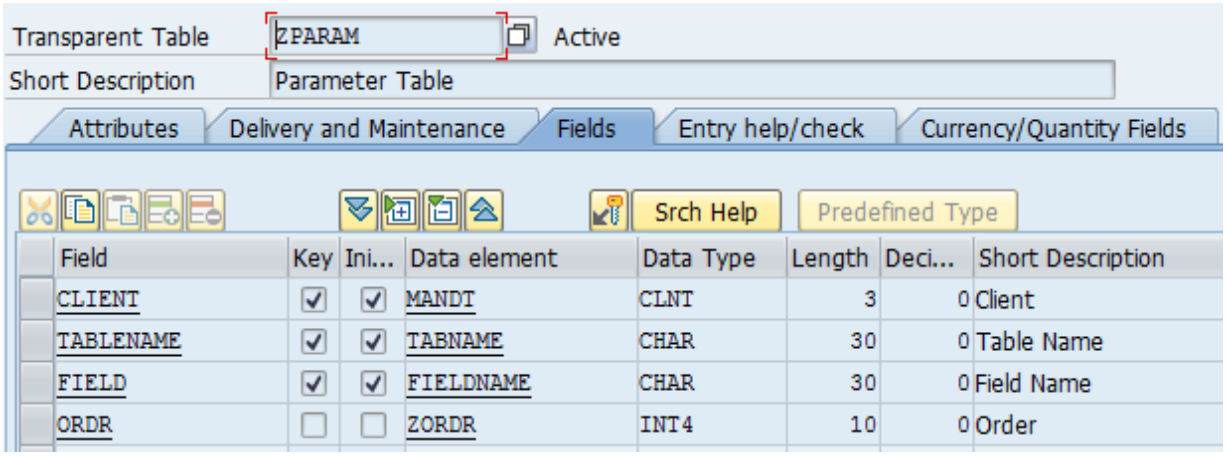

Fig. 1. Structure of the customizing table

The entries presented in Figure 2 exemplify the possible contents of the database table:

DOI: 10.12948/issn14531305/21.3.2017.05

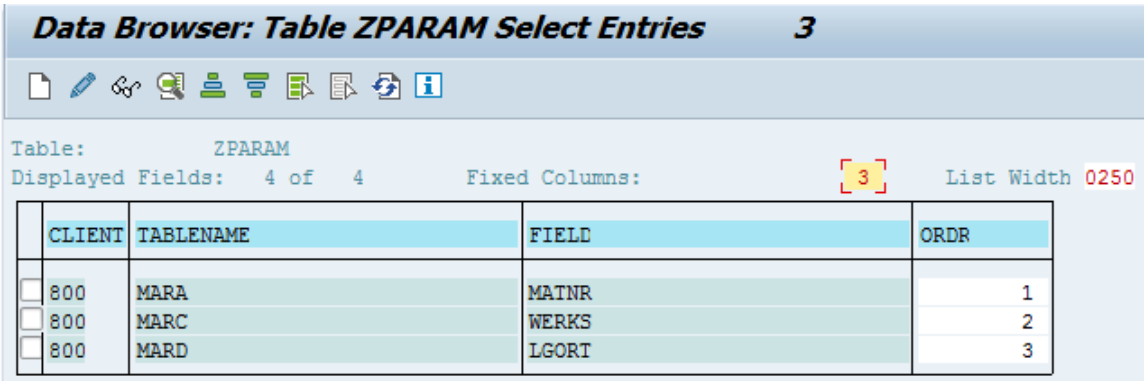

Fig. 2. Select entries of the database table

Based on these entries, we would like to have three select options on the dynamically built selection screen: first the **material number** (MARA-MATNR), second the **plant** (MARC-WERKS) and third the **storage location** (MARD-LGORT). We have chosen select options instead of parameters, as these offer more flexibility, like using intervals for example. Our proposed solution is to embed ABAP code in a variable dyn_sel_screen of type table c(100) and then submit this report via a SUBMIT … AND RETURN statement.
The main program ZADAPT has the purpose to generate the coding for our called program ZADAPT_CALLED. Therefore, it will contain the top include, for data declaration purposes, the SELECT statement for the data selection from our customizing table, and a lot of APPEND statements, which will build line by line the final program, that we need to submit. The general schema of the program could be summarized in the diagram shown in Figure 3.
The code lines with the general declarations of our application are presented in annex 1. In these lines of code the top include ZADAPTTOP will contain the general declarations of the program. Here, we have created a type called ty_abapsource of TYPE c LENGTH 100, and some variables, which use this type: dyn_sel_screen, code_line, lt_int_comment and ls_int_comment.

```
*&---------------------------------------------------------------------*
*& Report  ZADAPT
*&---------------------------------------------------------------------*
*& This report is intended for test purposes.
*& We would like to create a dynamic selection screen. Depending on
*& data found in a customizing table, we would like to generate a
*& program with a selection screen, containing select options based
*& on the criteria specified in the customizing table ZPARAM.
*&---------------------------------------------------------------------*
INCLUDE ZADAPTTOP                               .    " global Data
* INCLUDE ZADAPTO01                              .  " PBO-Modules
* INCLUDE ZADAPTI01                              .  " PAI-Modules
* INCLUDE ZADAPTF01                              .  " FORM-Routines
```

Other include files, as the PBO and PAI modules for example, have been commented, as our focus will be the dynamic selection screen generation, and not the further processing of data.

```
* Collect all information from the customizing table
  TRY.
    SELECT * FROM zparam
      INTO CORRESPONDING FIELDS OF TABLE lt_param
      ORDER BY ordr.
  CATCH cx_sy_sql_error.
    MESSAGE e460(zadapt_mess) RAISING selection_failed.
  ENDTRY.
```

DOI: 10.12948/issn14531305/21.3.2017.05

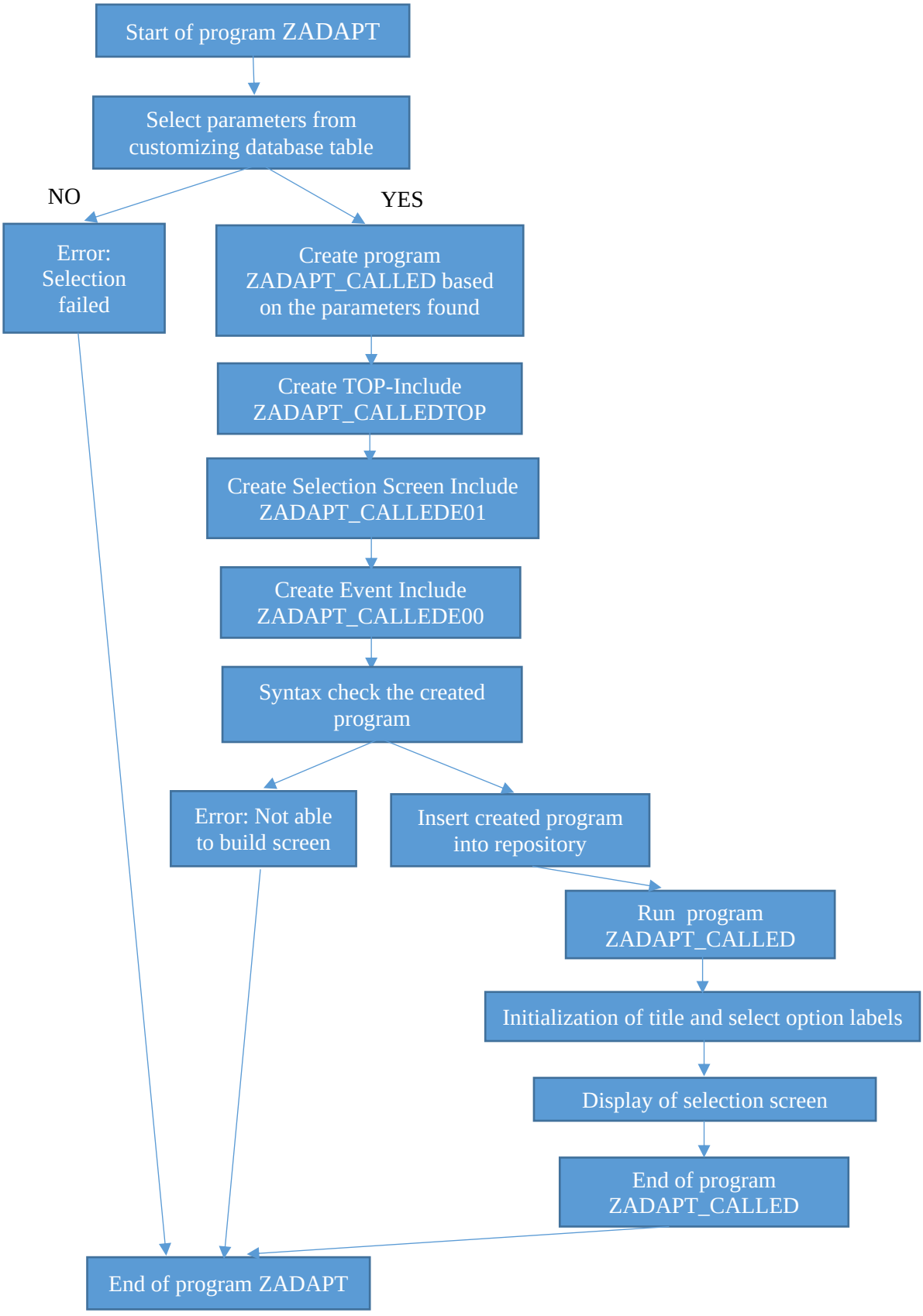

Fig. 3. Diagram of the elaborated program

The SELECT statement is embedded in a TRY-CATCH block in order to prevent short dumps due to a faulty selection process. We will select all the entries from the customizing

DOI: 10.12948/issn14531305/21.3.2017.05

table ZPARAM into a variable lt_param (TYPE STANDARD TABLE OF zparam). No WHERE clause is necessary, as no more than 1000 entries are expected to be available in the customizing table, and all entries have to be displayed in the dynamically generated selection screen. That's why the SELECT statement also does not have to be optimized for better performance. The ORDER BY clause has been added to ensure the sorting of the fields by the column ORDR of the database table.

```
* Introduction of the report to be defined
  APPEND '*&---------------------------------------------------------------*' TO dyn_sel_screen.
  APPEND '*& * This report was generated by the program ZADAPT.           ' TO dyn_sel_screen.
  APPEND '*& * Changes to the source text therefore have no effect.       ' TO dyn_sel_screen.
  APPEND '*& * Changes have to be made in the program ZADAPT              ' TO dyn_sel_screen.
  APPEND '*&---------------------------------------------------------------*' TO dyn_sel_screen.
```

The first step towards the creation of our dynamic selection screen is to add an explanation to the header of the called program, so the user who views the coding for the first time should know that the coding has been generated automatically by another program and that changing it will have no influence whatsoever. The program ZADAPT, which generated the program ZADAPT_CALLED has also been specified.

The further lines of code, presented in annex 2, define the top include of the called program ZADAPT_CALLED. As we observe, another variable than dyn_sel_screen has been used: sub_tab. That means, another include file has been created for the called program via the INSERT REPORT … FROM … PROGRAM TYPE 'I' statement.
Observation: Only after the COMMIT WORK statement will the include program be available in the repository also.

```
  APPEND ' INCLUDE ZADAPT_CALLEDTOP.                                    ' TO dyn_sel_screen.
  APPEND '                                                              ' TO dyn_sel_screen.
```

The next step is to declare the top include of the called program and to define the dynamically generated selection screen. The afferent code can be viewed in annex 3. In this section of code listed in annex 3, we will have a loop on the table lt_param, in which we have selected our data from the customizing table. A counter is needed (lv_count_parameter/lv_int_parameter) in order to ensure uniqueness of every declared SELECTION-OPTION. We have used in-line data declaration for every select option with the DATA statement, as we know the type only at runtime. These so_xxx will be initialized during the INITIALIZATION event of the program and with the help of the function of DDIF_FIELDINFO_GET we have access to the data element description (SCRTEXT_L) of each field of the customizing table.
If no entries are found in the customizing table (ELSE branch), a message will be displayed, stating to maintain the customizing table.

Afterwards, the include ZADAPT_CALLEDE01 corresponding to the dynamic selection screen generation is inserted into the repository also.
In the initialization event (see line codes presented in annex 4), the title of the program is passed, along with the types and descriptions of the select options (stored in the lt_int_comment variable).
Before running the dynamically generated report, a syntax check is necessary. If errors are found, a message will be displayed to the user, in order to contact the system administrator. Otherwise, the generated program ZADAPT_CALLED will be executed. This section of code can be seen in annex 5.
The program ZADAPT_CALLED will look like that shown in annex 6.
The selection-screen ZADAPT_CALLEDE01 is presented in annex 7.

DOI: 10.12948/issn14531305/21.3.2017.05

The code lines of the initialization include-program ZADAPT_CALLEDE00 are presented in annex 8.

As a result, the generated selection-screen, represented in Figure 3, will have no more and no less than the desired selection fields, specified in the customizing table ZPARAM:

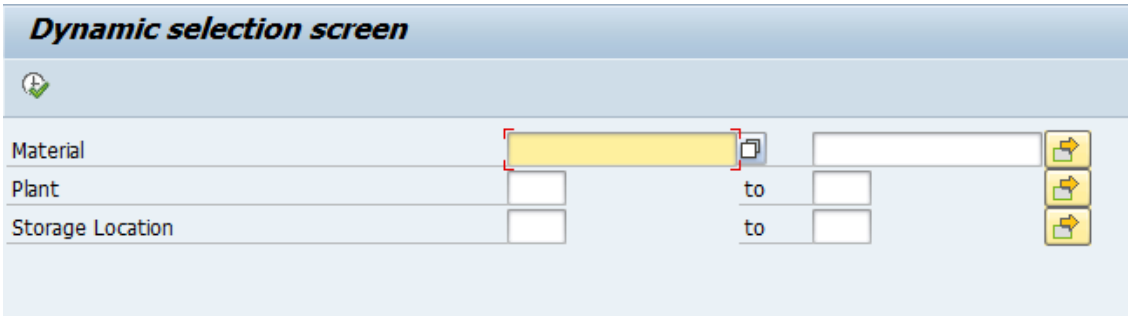

Fig. 3. The generated selection screen

As expected, the select options available on the generated selection screen correspond to the data maintained in the customizing table.

4 Advantages of the proposed solution

The proposed solution for the dynamic selection screen generation has a lot of advantages. We will name just a few, in order to convince our readers:

- **We don't have to rewrite the program**, in order to define additional selection parameters or select options. The simple solution of adding an entry in a customizing table is more adequate than the changing the whole coding of the program, in order to add a parameter in the whole logic of the program. In some cases not only the top include may have to be adapted, but also the selection process in a lot of places within the whole program. With a dynamic approach, this is simplified.
- Each customer could **define various selection-screen versions** for each client, as the customizing table is client-dependent. By logging in with another client, the program may be configured to display certain select options for certain clients. This would not be possible if the customizing table would not be client dependent and certainly not for the manual modification of the coding itself.
- The customer can **configure the selection fields in the desired order**. This is also an advantage, as many customers have a desired order, in which they would like the fields to be displayed. Only the change of the select options order would take some time to modify the coding. The usage of the customizing table, regarding this point, comes in handy, solving a lot of unnecessary code change.
- Generally speaking, programs and subroutines with dynamic features are much **more flexible and generic**.

On the other hand, the dynamic approach comes with a handful of shortages:

- Although the initial development of a dynamic solution is **a bit more time-consuming**, on the long run it proves to be quite a good alternative to classic screen-programming in ABAP, as no further coding is needed, only customizing entries in a database table.
- Depending on the size of the data maintained in the customizing tables, a **moderate increase in storage space** is expected in comparison to a static approach. However, the expected increase in storage space will not have any negative impact on the performance of the program.
- Programs and subroutines with dynamic features can become **more complex and harder to maintain**.

5 Conclusions

As shown with the help of the coding blocks above, which help us understand the idea of a

DOI: 10.12948/issn14531305/21.3.2017.05

dynamic selection screen generation in the context of ABAP programming, the proposed solution provides a modern approach, which may be used in other programming languages and graphical user interfaces as well. Every program interacts with the user in one way or another and may use selection screens, which should be customizable, in order to ensure productivity and avoid confusion among the end users regarding unnecessary and unused parameters.

The purpose of simplifying the user interface of a program and adjust it accordingly, so only the desired or required parameters/select options are displayed, has to be on the mindset of every developer, as the end user has a final say about the program and he may influence supervisor, advising the use of a competing solution. In this context, if a short delay in the development process is not critical and a moderate storage increase can be tolerated, we can conclude that it is advisable to use dynamic programming techniques in order to achieve a more flexible program, which can be adapted according to customer needs without writing additional program coding.

References

[1] Dynamic Programming, Available : https://en.wikipedia.org/wiki/Dynamic_programming [Accessed Mai 2017].

[2] C. Anthony, "ABAP Dynamic Programming – Part 1", 16 February 2015. [Online]. Available: http://www.itpsap.com/blog/2015/02/16/abap-dynamic-programming-part-1/ [Accessed March 2017]

[3] C. Anthony, "ABAP Dynamic Programming – Part 2", 15 March 2015. [Online]. Available: http://www.itpsap.com/blog/2015/03/15/abap-dynamic-programming-part-2/ [Accessed March 2017]

[4] SAP Documentation, Dynamic Programming, [Online]. Available: https://help.sap.com/saphelp_nw74/helpdata/en/f7/2ca4e7ac3b471182783bf8540b0a0a/content.htm [Accessed February 2017]

[5] K. Mithun, Programming for Selection Screens with Dynamic Properties, [Online]. Available: https://blogs.sap.com/2012/08/17/dynamic-screen-programming-for-selection-screens/ [Accessed March 2017]

Cornelia L. MUNTEAN has graduated the Faculty of Computer Science at the „Politehnica" University of Timişoara in 1986. She holds a PhD diploma in Engineering and is currently assistant professor at the department of Business Information Systems at the West University of Timişoara. She joined the staff of the Faculty of Economics and Business Administration of the West University of Timisoara in 1992 as a teaching assistant, then graduated in 1997 as a senior lecturer and in 2005 as an assistant professor. She is the author of 7 books and about 60 journal articles in the field of artificial intelligence, intelligent systems and programming.

DOI: 10.12948/issn14531305/21.3.2017.05

Annexes

Annex 1:

```
*&---------------------------------------------------------------------*
*& Include ZADAPTTOP                                         Report ZADAPT
*& Data declaration include for ZADAPT Program
*&---------------------------------------------------------------------*
REPORT ZADAPT.

* Report name
  DATA: lv_rep              LIKE sy-repid.

  TYPES:
*      line of coding
      ty_abapsource         TYPE c LENGTH 100.

  DATA:
*     Variables to read customizing settings
        lt_param            TYPE STANDARD TABLE OF zparam,
        ls_param            TYPE zparam,
        gt_dfies            TYPE TABLE OF dfies,
        gs_dfies            TYPE dfies,

*     Generic Select-Option name
        so_name(80)         TYPE c,

*     Run the generic coding
        prog                TYPE c, " Name of the submitted program
        dyn_sel_screen      TYPE TABLE OF ty_abapsource, " final coding of the new
program
        code_line           TYPE ty_abapsource, " code line
*     Message for syntax error
        mess                TYPE string,
        lin                 TYPE i,    " row where the error occured
        wrd                 TYPE c LENGTH 8,   " word, where error occured

*     Creation of subroutines dynamically
        sub_prog            LIKE sy-repid,
        sub_tab             TYPE STANDARD TABLE OF string,
        sub_mess            TYPE string,
        sub_sid             TYPE string,

        lv_int_parameter    TYPE string, " Necessary for the creation of unique Par
ameter descriptions
        lv_count_parameter  TYPE i VALUE 0, " Counts the number of parameters on sc
reen --> used as unique parameter identifier
        lv_parameter_type   TYPE string,

        lt_int_comment      TYPE TABLE OF ty_abapsource,
        ls_int_comment      TYPE ty_abapsource. " Comment for field
```

Annex 2:

```
 * Data declaration
   APPEND '                                                                ' TO sub_tab.
   APPEND 'REPORT ZADAPT_CALLED .                                          ' TO sub_tab.
   APPEND '                                                                ' TO sub_tab.
   APPEND 'DATA: title(80) TYPE c.                                         ' TO sub_tab.
   APPEND 'SET TITLEBAR title OF PROGRAM sy-repid .                        ' TO sub_tab.
   APPEND '                                                                ' TO sub_tab.
   APPEND '************************************************************************' TO sub_tab.
   APPEND '* Selection-screen                                              ' TO sub_tab.
   APPEND 'INCLUDE ZADAPT_CALLEDE01.                                       ' TO sub_tab.
   APPEND '************************************************************************' TO sub_tab.

   sub_prog = 'ZADAPT_CALLEDTOP'.
   INSERT REPORT sub_prog FROM sub_tab PROGRAM TYPE 'I' .
   COMMIT WORK.
   REFRESH sub_tab.
```

DOI: 10.12948/issn14531305/21.3.2017.05

Annex 3:

```
APPEND '* Selection screen with parameters from                                  ' TO sub_tab.
  IF lt_param IS NOT INITIAL.
    LOOP AT lt_param INTO ls_param.
      lv_count_parameter = lv_count_parameter + 1.
      lv_int_parameter = lv_count_parameter.
      CONDENSE lv_int_parameter.
      APPEND '                                                                   ' TO sub_tab.
      APPEND '  SELECTION-SCREEN BEGIN OF LINE.                                  ' TO sub_tab.
      CONCATENATE '    SELECTION-SCREEN COMMENT 1(37) s' lv_int_parameter ' FOR FIELD s_'
        lv_int_parameter ' .' INTO code_line.
      APPEND code_line TO sub_tab.
      CLEAR code_line.
      CONCATENATE ls_param-tablename '-' ls_param-field INTO so_name.
      CONCATENATE '    DATA so_' lv_int_parameter '(80) TYPE c.' INTO code_line.
      APPEND code_line TO sub_tab.
      CLEAR code_line.
      CONCATENATE '    SELECT-OPTIONS s_' lv_int_parameter INTO code_line .
      CONCATENATE code_line ' FOR (so_' lv_int_parameter ') .' INTO code_line .
      APPEND code_line TO sub_tab.
      CLEAR code_line.
      APPEND '  SELECTION-SCREEN END OF LINE.                                    ' TO sub_tab.
      CALL FUNCTION 'DDIF_FIELDINFO_GET'
        EXPORTING
          tabname              = ls_param-tablename
          LANGU                = SY-LANGU
        TABLES
          DFIES_TAB            = gt_dfies
        EXCEPTIONS
          NOT_FOUND            = 1
          INTERNAL_ERROR       = 2
          OTHERS               = 3 .
      READ TABLE gt_dfies WITH KEY fieldname = ls_param-field INTO gs_dfies.
      CONCATENATE '  s' lv_int_parameter ' = ''' gs_dfies-
scrtext_l ''' . ' INTO ls_int_comment.
      APPEND ls_int_comment TO lt_int_comment.
      CONCATENATE '  so_' lv_int_parameter ' = ''' gs_dfies-tabname '-' gs_dfies-
fieldname ''' .'
        INTO ls_int_comment.
      APPEND ls_int_comment TO lt_int_comment.
    ENDLOOP.
  ELSE.
    MESSAGE e470(zadapt_mess) DISPLAY LIKE 'I'.
  ENDIF.

  sub_prog = 'ZADAPT_CALLEDE01'.
  INSERT REPORT sub_prog FROM sub_tab PROGRAM TYPE 'I' .
  COMMIT WORK.
  REFRESH sub_tab.
```

Annex 4:

```
  APPEND '* INITIALIZATION Event                                    ' TO dyn_sel_screen.
  APPEND ' INCLUDE ZADAPT_CALLEDE00.                                ' TO dyn_sel_screen.
  APPEND '                                                          ' TO dyn_sel_screen.

* INITIALIZATION
  APPEND '                                                          ' TO sub_tab.
  APPEND 'INITIALIZATION.                                           ' TO sub_tab.
  APPEND '                                                          ' TO sub_tab.
* Title
  CONCATENATE '  sy-title = ''' text-tit '''. ' INTO code_line.
  APPEND code_line TO sub_tab.
  APPEND '                                                          ' TO sub_tab.
  APPEND LINES OF lt_int_comment TO sub_tab.

  sub_prog = 'ZADAPT_CALLEDE00'.
  INSERT REPORT sub_prog FROM sub_tab PROGRAM TYPE 'I' .
  COMMIT WORK.
  REFRESH sub_tab.
```

DOI: 10.12948/issn14531305/21.3.2017.05

Annex 5:

```
* Syntax check
  SYNTAX-CHECK FOR dyn_sel_screen MESSAGE mess LINE lin WORD wrd PROGRAM prog.

  IF sy-subrc = 4 OR sy-subrc = 8.
**   Not able to build screen. Please contact your system administrator!
    MESSAGE w480(zadapt_mess).
  ELSE.
    lv_rep = 'ZADAPT_CALLED'.
*     Insert the program into the repository
    INSERT REPORT lv_rep FROM dyn_sel_screen.
*     Only after the Commit work statement is the program in the repository.
    COMMIT WORK.
*     Executes the generated coding to create the selection screen.
    SUBMIT (lv_rep) VIA SELECTION-SCREEN AND RETURN.
  ENDIF.
```

Annex 6:

```
*&---------------------------------------------------------------------*
*& * This report was generated by the program ZADAPT.
*& * Changes to the source text therefore have no effect. Changes have
*& * to be made in the program ZADAPT!
*&---------------------------------------------------------------------*
 INCLUDE ZADAPT_CALLEDTOP.

* INITIALIZATION Event
 INCLUDE ZADAPT_CALLEDE00.
The top include contains the following lines of code:
REPORT ZADAPT_CALLED .

DATA: title(80) TYPE c.
SET TITLEBAR title OF PROGRAM sy-repid .

************************************************************************
* Selection-screen
INCLUDE ZADAPT_CALLEDE01.
************************************************************************
```

DOI: 10.12948/issn14531305/21.3.2017.05

Annex 7:

```
* Selection screen with parameters from

  SELECTION-SCREEN BEGIN OF LINE.
    SELECTION-SCREEN COMMENT 1(37) s1 FOR FIELD s_1 .
    DATA so_1(80) TYPE c.
    SELECT-OPTIONS s_1 FOR (so_1) .
  SELECTION-SCREEN END OF LINE.

  SELECTION-SCREEN BEGIN OF LINE.
    SELECTION-SCREEN COMMENT 1(37) s2 FOR FIELD s_2 .
    DATA so_2(80) TYPE c.
    SELECT-OPTIONS s_2 FOR (so_2) .
  SELECTION-SCREEN END OF LINE.

  SELECTION-SCREEN BEGIN OF LINE.
    SELECTION-SCREEN COMMENT 1(37) s3 FOR FIELD s_3 .
    DATA so_3(80) TYPE c.
    SELECT-OPTIONS s_3 FOR (so_3) .
  SELECTION-SCREEN END OF LINE.
```

Annex 8:

```
INITIALIZATION.

  sy-title = 'Dynamic selection screen'.

  s1 = 'Material' .
  so_1 = 'MARA-MATNR' .
  s2 = 'Plant' .
  so_2 = 'MARC-WERKS' .
  s3 = 'Storage Location' .
  so_3 = 'MARD-LGORT' .
```

DOI: 10.12948/issn14531305/21.3.2017.05