CODE-GENERATION AS THE EFFECTIVE TOOL OF ADAPTATION OF PROGRAM PRODUCTS FOR BUSINESS-PROCESSES OF ENTERPRISES

[a]ILYA PLESHCHINSKII, [b]NIKOLAI PLESHCHINSKII

Kazan Federal University, 18 Kremlyovskaya street, Kazan 420008, Russian Federation, Russia
e-mail: [a]editor@ores.su, [b]pnb@kpfu.ru

Abstract: In the work one way to adapt boxed software products for the business processes of enterprises is considered. A method of implementing scenario in the form of graphical block diagrams is proposed. User creates a visual flowchart of the process. Then on the basis of this flowchart the code is automatically generated and executed. scenario variables are transformed into local variables, each component of the flowchart represents a call to a function, and expressions become classical language expressions. It is valid and effective to use the *goto* operator to ensure logic of transitions between components. A scenario of automatic service of contact center for reception of meter in text chat is considered as an example. Despite the relatively labor-intensive development of such a code generator, debugging is reduced to checking the correctness of the resulting code. Adding new components requires no modification of the basic module of a code generator. Performance of such scripts is maximal and practically does not differ from changes made in the source code of the software product.

Keywords: code-generation, adaptation of program products, business-process of enterprise

1 Introduction

Currently, it is impossible to imagine the work of the enterprise of any level without software products. Even small companies use in their daily work those or other means for the conduct of the customer base and the history of relations with them, automate the processing of appeals through various channels of communication, introduce modern technologies of automatic and semi-automatic service. Representatives of medium and large business usually solve these problems more effectively. They use the unified corporate information systems (Oleinik, 2012; Oleinik, 2001), or arrange the close integration of several specialized software products. All software solutions for business can be divided into two categories:

- solo (developed for the specific needs of one enterprise and initially taking into account all the specifics of its processes);
- boxed (universal, designed for a sufficiently broad business industry and involving adaptation for the specifics of concrete enterprise during the implementation phase).

This article focuses on adaptation technologies of boxed software products for the business processes of enterprises (Fedorova, 2016). The scope of a software product in this case has no special meaning. It can be a classic CRM (customer relationship management system), contact center (call center for the reception and processing both phone calls and referrals via e-mail, mobile messenger, chat, social networks, etc.), Service Desk (ticket system of technical support service), ERP (enterprise resource planning).

Developer of the boxed software always is teetering between the two extremes. On the one hand, it is desirable to establish a universal and flexible product that can solve problems of any enterprise. On the other hand, the modern market needs simple products that do not require significant implementation costs on the introduction and adaptation (Sneller & Lineke, 2014; Okriashvili, 2017)

2 Different approaches to adaptation technologies

In some cases, adaptation of the software product is limited to configuring settings and filling manuals. For example, it can be the nomenclature of goods and services, category of clients, social networking accounts and email for automatic processing of the application, etc. Design of system parameters and implementing of custom handbooks is fairly simple and is not considered in this article.

A change of the logic of the software product, depending on the specific business processes of the enterprise, is the most interesting to us. An important condition is the independence from the developers, i.e. we do not need to change the original product codes and to assembly its new version with the participation of the developer.

From the user's point of view, there are three approaches to adapting the software product for business processes:

- scripts (code written on one of the programming languages);
- special configuration files (typically, prepared on the basis of the XML format to suit the specific requirements of the developer);
- graphic scenarios (representations of algorithms in the graphical editor in the form of block diagrams) (Scienna, 2011).

Practice shows that the most convenient for the user is the third approach. You can select the obvious advantages of graphics scenarios:

- graphical block diagrams are clear and intuitive;
- creating and modifying block diagrams does not require the skills of a programmer;
- the cost of training new specialists and transfer to them the tasks of the development and support of graphic scenarios are minimal.

Further, we consider one way to implement the scenario in the form of graphical block diagrams, as well as two approaches to their subsequent execution. As an example, scenario of automatic service contact center that accepts the testimony of one tariff energy meter in the text chat (online consultant on site, mobile messenger or social page network) will be considered.

3 Formalization of algorithmic scenarios

In programming terms, the scenario (script) is a function that is called some event occurs (for example, when an incoming phone call takes place, when you receive an email, when you change the status of the client, etc.). In our case, the script runs when a new message arrives from the user. The script function can have a set of input parameters, can refer to some of the objects of the software product, and, sometimes, can return a result. The script can be performed long enough (for example, if it contains the call of waiting functions), so normally it should be performed in a separate thread. This allows you to avoid locking the modules of the system for the all script execution time. In our example, the script can be executed for two minutes, because the user is given twice the time to enter information (Herlihy & Shavit, 2012; Villalobos Antúnez, 2016).

The scenario is a block diagram consisting of interconnected components directed transitions. It means that the component performs some actions and then the script executing on one of the transitions continues.

The script has one entry point («Start» component) and completes by the component «Stop» or «Return the result». Generally speaking, these components can be as many as you want.

To provide basic logic, components «Condition» and «Options» are provided that allow scenario branching depending on those or other conditions. Also scenario branching can be implemented when there are multiple transitions from a single component (for example, if a «success» occurs or «failed» when you run a SQL-query).

To work with variables, components «Declare the variable» and «Assign the value» are used. In the same way the component «Cycle» can be implemented, although it's just enough to

organize it by combination of components «Condition» and «Assign the value».

Components for integration with third-party services like «SQL-query» and «HTTP-query» enable you to perform any queries for further use of the result obtained in the scenario.

Other components depend on the purpose of the scenario. For text message processing scenario three components are used: «Reply», «Wait for the message» and «Put the message in the queue to the operators». For the scenario of interactive voice menu there are significantly more components usually. The main of them are: «Pick up», «Hang up» «Play file», «Play value», «Record a voice message», «Speech synthesis», «Switch to the number», «Speech recognition», «Input DTMF-data», «Switch to the number», «Switch to turn operators».

An important part of any script is the «Expression». The expressions can be passed as parameters to components, and it is possible to assign variables with their values. The main types of the expressions are the following: a constant, a variable, a parameter of a script, an operator.

In boxed software product developer implements the module « scenario Editor». This module enables the user the following options:

- to create scenarios
- to generate visually scenarios as algorithmic block diagrams
- to check scenarios for errors
- to assign scenarios to run automatically upon the occurrence of various events of the software product

Technology realization of graphical scenario editor, as well as data storage format can be absolutely arbitrary. It makes sense to develop a scenario editor, guided by State standards (in Russia) or generally accepted notations (Herrera, 2015).

The scenario of reception of meters in this formalization is shown in the figure.

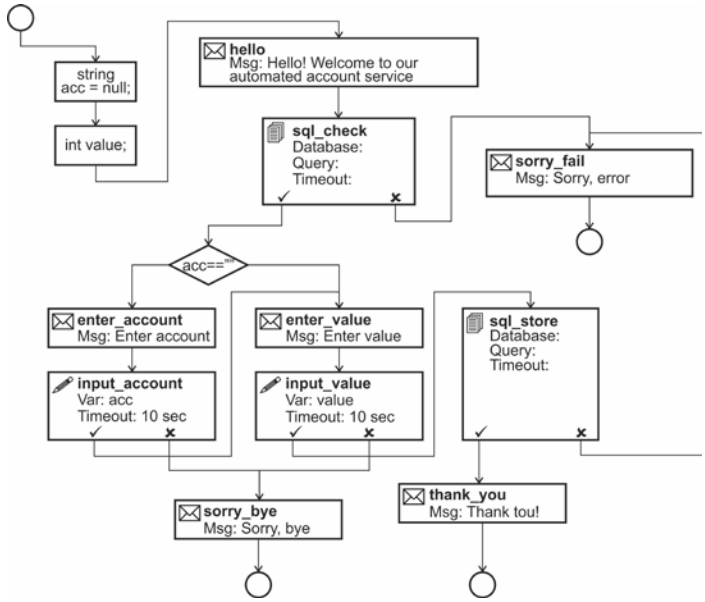

Fig 1. Script of receipt of counter indications

4 A simple interpretation of the scenarios

To interpret scenarios, developer of software product implements the module «Interpreter». This module loads the scenario into memory and executes the components, manages the contents of variables, provides logic of conditional operators and cycles, calculates the values of expressions, etc. This approach has the following disadvantages:

- labor expenditures on developing and debugging this interpreter are very high;
- adding new components into scenario editor would require finalization of the interpreter;
- performance will be quite low;
- the overall functionality of extensibility of scripts is also low.

5 The conversion of scenario in code

Taking into account the scenario is a function, we offer a different approach. The software product developer implements the module «Code generator». This module based on graphical scenario generates the code for the function (Hack et al, 2017). Depending on the platform and programming language used in the development of a software product, code may require compilation (for example, for languages (Stroustrup, 2014) C++ or C# (Troelsen & Japikse, 2015)), or immediately be passed to the interpreter.

When code generation, script variables are transformed into local variables, where each component represents a call to a function, and expression becomes classical language expressions. We believe that it is valid and effective to use the goto operator to ensure logic of transitions between components.

The example of code generated by the scenario of the counter reception, in C#:

```
public void RunScenario(MessengerScenarioCore Core, string Message, string IDClient)

{
   string Account = null;
   int Value = 0;
   goto label_hello_103;
label_hello_103: ;
   Core.SendMessage("Hello! Welcome to our automated account service");
   if (Core.SQLQuery("Select account from data where idclient=%1 order by time desc limit 1", IDClient, out Account))
      goto label_check_account_131;
   else
      goto label_sorry_fail_121;
label_sorry_fail_121: ;
   Core.SendMessage ("Sorry, error. Try later")
   return;
label_check_account_131: ;
   if (Account == "")
```

```
      goto label_answer_enter_account_153;
   else
      goto label_answer_enter_value_166;
label_answer_enter_account_153: ;
   Core.SendMessage ("Enter account");
   if (Core.WaitForMessage(60 * 1000, out Account))
      goto label_answer_enter_value_166;
   else
      goto label_sorry_bye_168;
label_answer_enter_value_166: ;
   Core.SendMessage (("Enter value" + Account));
   goto label_ZHdat_soobshhenie2_198;
label_sorry_bye_168: ;
   Core.SendMessage("You did not input data. Bye!");
   return;
label_ZHdat_soobshhenie2_198:;
   if (Core.WaitForMessage(60 * 1000, out Value))
      goto label_sql_store_201;
   else
      goto label_sorry_bye_168;
label_sql_store_201: ;
   if (Core.SQLQuery("insert into data (account, idclient, value,
time) values (%1, %2, %3, now())", Account, IDClient, Value))
      goto label_answer_thank_you_203;
   else
      goto label_sorry_fail_121;
label_answer_thank_you_203: ;
   Core.SendMessage ("Thank you! ") ;
   return;
}
```

Despite the relatively labor-intensive development of such a code generator, debugging is reduced to checking the correctness of the resulting code. Adding new components requires no modification of the basic module of a code generator; because it is enough just to put a call to the desired function in compliance with the component. Performance of such scripts is maximal and practically does not differ from changes made in the source code of the software product.

In addition, code generation technology allows you to add into script editor code component «Code fragment». This allows you to not only to complete the scenario with any functionality not provided in the form of components, but also completely replace the graphical block diagram. Thus, we have combined two previously described approaches to adaptation (the first and the third).

Sample code to implement identical script in C# without using graphical block diagram:

```
public void RunScenario(MessengerScenarioCore Core, string
Message, string IDClient)

{
   string Account = null;
   int Value = 0;
   Core.SendMessage("Hello! Welcome to our automated
account service");
   if (Core.SQLQuery("select account from data where
idclient=%1 order by time desc limit 1", idclient, out Account))
   {
      if (Account == "")
      {
         Core.SendMessage("Enter account");
         if (!Core.WaitForMessage(60 * 1000, out Account))
         {
            Core.SendMessage("You did not input data. Bye!");
            return;
         }
      }
      Core.SendMessage(("Enter value" + Account));
      if (Core.WaitForMessage(60 * 1000, out Value))
      {
         if (Core.SQLQuery("insert into data (account, idclient,
value, time) values (%1, %2, %3, now())", Account, IDClient,
Value))
            Core.SendMessage("Thank you!");
         else
            Core.SendMessage("Sorry, error. Try later");
      }
      else
         Core.SendMessage("You did not input data. Bye!");
   }
   else
      Core.SendMessage("Sorry, error. Try later");
}
```

6 Summary

The code that you write manually, of course, is more clear and understandable. But writing it requires a highly skilled programmer. But the code generated by code generator, despite the complexity and redundancy of unconditional transitions goto provides identical functionality and performance.

7 Conclusions

The authors participated in the development of several software products, in which the graphic scenarios have been applied. In the first generation of products the second approach was applied, and in subsequent generations the third approach was applied. According to our estimates, the labor expenditures on developing and debugging were reduced by more than twice, and performance has increased.

Application of code generation is not limited to the execution of graphical block diagrams. Code generation can be applied for symbolic computation, to build a comfortable and productive object models (ORM) or even for realizing approach “program that improves itself”.

Acknowledgements

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

Literature:

1. Oleinik P.P. (2012). Corporative information systems, Sankt-Petersburg: Piter.
2. Oleinik P.P. (2001). Basic standards of the corporative information systems, LAP Lambert Academic Publishing.
3. Fedorova G.N. (2016). Development, embedding and adaptation of program soft of industry orientation, Moscow, KURS.
4. Sneller R.C., Lineke, A.(2014). Guide to ERP. Benefits, Implementation and Trends. London: Bookboon.
5. Scienna S. (2011). Algorithms, Sankt-Petersburg. BHV Petersburg.
6. Herlihy M., Shavit N., (2012). The Art of Multiprocessor Programming, Morgan Kaufmann, 2012.
7. Herrera E., (2015). The BPMN Graphic Handbook. CreateSpace Independent Publishing Platform.
8. Hack S., Wilhelm R., Seidl H. (2017). Compiler Design Code Generation and Machine-Level Optimization. Springer-Verlag Berlin Heidelberg.
9. Stroustrup B. (2014). C++ Programming Language: C++ 11, Addison Wesley Longman.
10. Troelsen A., Japikse P. (2015). C# 6.0 and the .NET 4.6 Framework. APress.
11. Okriashvili T.G. (2017). The State of Private Law in the Modern Legal Society, Astra Salvensis, Supplement No. 2, p. 539.
12. Villalobos Antúnez J.V. (2016). Ciencia y Tecnología para la libertad, Opcion, 32(79), pp. 7-9.