A model-Driven Approach to User-Specific Operations in the Development of Web Business Applications

Vladimir Balać*, Milan Vidaković**
* Faculty of Technical Sciences/Computing and Control Engineering, Novi Sad, Serbia
** Faculty of Technical Sciences/Computing and Control Engineering, Novi Sad, Serbia
vlad82@uns.ac.rs, minja@uns.ac.rs

***Abstract*—Operations over data represent the main functionality of web business applications. User-specific operations extend the set of common create, read, update, and delete (CRUD) operations. As user-specific operations require an underlying algorithm definitions, they are not as generic as CRUD operations are. By this, user-specific operations do not have an extensive model-driven development support and the abstract models of existing model-driven tools do not support such algorithms by default. Without a proper development support, developers use custom manual workarounds to develop user-specific operations. This paper proposes a new model-driven approach that supports user-specific operation definitions through visual algorithm modeling. The approach also supports an automated code writing from the aforementioned algorithm model. The proposed approach supports unified declaration of the same user-specific operation across different platforms. By this, the same operations would not have a different declaration for different platforms and a portability of operations would not require an extra implementation effort.**

I. INTRODUCTION

Web business applications are data-intensive and deal with various operations over data. The create, read, update, and delete (CRUD) operations are common for all data-intensive applications. By this, all CRUD related application features are suitable for the code generation. Such generic nature allows model-driven tools to have an extensive CRUD development support. Model-driven tools shift the focus of application development from code writing to higher-level abstract modeling [1, 2]. The model represents a formal, complete, and consistent abstraction of an application, as stated in [3], from which code generation facilities write the target platform code. We consider a platform as any set of concrete development tools in any programming language.

Business web applications may also provide a set of user-specific operations. User-specific operations are unique for each application and deal with the requirements that CRUD operations do not cover. Calculating cars fuel efficiency or orders total price are examples. Such operations require an algorithm description, i.e. a self-contained systematic set of expressions such as declaring variables and assigning values. By this, user-specific operations are not as generic as CRUD operations are.

User-specific operations do not have the same model-driven development support as CRUD operations do. By this, the abstract application models do not define user-specific operations. Model-driven tools do not support user-specific algorithm modeling. Such modeling is a necessary development step and tools ought to support it. Without the support, developers use custom manual workarounds outside of the model-driven approach. The workarounds take place after the code generators produce the application core features. Such approach makes portability of user-specific operations a difficult process. The same operations have different definitions and implementations for different platforms.

In this paper, we have addressed the problem of a model-driven support for user-specific operations through several steps. In Section 2, we have pointed out the lack of the support in the chosen representative model-driven and traditional tools. Following in Section 3, we have proposed a new model-driven approach that supports a platform-independent specification of a user-specific operation algorithm. The proposed specification takes the form of a customized flowchart diagram for algorithm description. A flowchart support shifts the focus on algorithm concepts rather than the underlying programming language syntax. A flowchart algorithm modeling supports four algorithm steps - declaration of variables, assignment of values, if conditions and while loops. The proposed flowchart support is part of the OdinModel framework [4] for the web business application development. The framework's code generation facilities support automated implementation of user-specific operations, which we have illustrated through an example in Section 4. At last, in Section 5, we resonate about proposed solution benefits and future work directions.

II. RELATED WORK

For our research, we have chosen representative tools that support user-specific operation implementation. To our knowledge, there are only two model-driven tools that support platform-independent algorithm specification by default. The first one is WebDSL, which supports algorithm definition through internal domain-specific language (DSL) [5]. By manual DSL code writing, developers may define all steps of an algorithm. Comparing WebDSL with our proposed approach, we may say WebDSL is locking developers to its native textual DSL. The OdinModel flowchart represents algorithms through visual diagram which is Unified Modeling Language (UML) alike. By this, our approach

may be more intuitive than WebDSL and more comprehensible to developers with lower-level development skills. The second found model-driven tool implements a similar approach to WebDSL. IIS*CASE uses IIS*CFuncLang DSLas to extend the internal DSL for algorithm code writing [6]. By this, IIS*CFuncLang supplements a basic CRUD set. The difference between IIS*CFuncLang with WebDSL and OdinModel is in target code implementation level. IIS*CFuncLang implements the code on the database platform level while the latter two implements the code on the application level. By this, the OdinModel framework supports an alternative approach which may be more suitable for developers that do not develop on the database level.

As for the development tools that do not use model-driven approach, we have found six tools that support user-specific operation modeling. The following five tools are implementing algorithm steps through a flowchart diagram: Flowgorithm [7], Larp [8], Raptor [9], Visual logic [10], and DRAKON Editor [11]. Flowgorithm, Raptor, and DRAKON produce and execute code of the flowchart, while Larp and Visual logic do not support code generators. The sixth tool is Blockly which is implementing algorithm steps in a puzzle form [12]. Blockly also produces the algorithm code from such puzzles. The crucial difference between aforementioned tools and our OdinModel framework is a data model support. Support for data model allows sharing of data between operations, databases, and object-oriented models. Neither one of the six presented tools supports a connection to an underlying data model.

III. SOLUTION OVERVIEW

The Flowgorithm tool is the main inspiration for our proposed flowchart approach. Flowgorithm supports creating multi-platform programs through flowcharts absent data models. By expanding the Flowgorithm concept, we have implemented and integrated our own flowchart within the OdinModel framework. By this, we have extended the framework's visual editor and the code generation facilities. Supporting both CRUD and user-specific operations in a single model is our main goal.

As stated in [4], the main part of the OdinModel specification is the Odin meta-model. The meta-model is a part of the framework's four-layered architecture. Such architecture is implementing Meta-Object Facility (MOF) - a standard for a DSL definition through an abstract meta-model [2, 13, 14]. The Odin meta-model covers concepts, such as entities, entity fields, relationships, and the steps of the user-specific operation algorithms. Fig. 1 shows the part of the meta-model which represents meta-classes for user-specific operation specification. For now, the Odin meta-model supports a declaration of variables, assignment of values, if conditions, and while loops.

The meta-class *CustomMethod* defines general attributes of each user-specific operation. The meta-class *DeclareVariable* specifies a declaration box - a part of the operation for variable declarations. The meta-class *AssignValue* specifies an assignment box inside which the existing variables from a declaration box receive values. The meta-class Value specifies a single assignment. Assigned value may be in a form of a mathematical

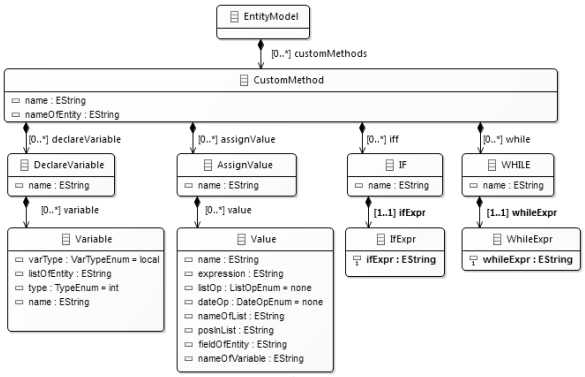

Figure 1. The user-specific operation meta-classes

expression, a value of another existing variable, or a variable that represents list values. The meta-class *IF* specifies a condition box that defines a logical expression of the condition and the subsequent operation expressions. Each condition box has only one logical expression defined through meta-class *IfExpr*. The OdinModel framework, for now, supports the while loop which executes code as long as a condition has the value true. The meta-class *WHILE* specifies a loop box that defines a logical expression of the loop and the subsequent operation expressions. Each loop box has only one logical expression defined through meta-class *WhileExpr*.

The Odin meta-model defines control flow – the rules that secure the order of expressions in a user-specific operation. The relations between the declarations, assignments, conditions and loops meta-classes define such rules. To sum it up, a declare box must be before an assignment box, and after logical expressions of a condition and a loop box. An assignment, condition, and loop box have an interchangeable order.

For the appropriate visual flowchart editor development, we have used Eclipse Sirius [15]. Sirius is a tool for custom graphical workbench creating. To build the appropriate code generators, we have used the Acceleo [16]. Acceleo is implementing the MOF model to text language standard.

IV. USE-CASE EXAMPLE

Application developers model a user-specific operation using a visual editor for the Odin model. Code generators produce the complete executable code from that model. A visual editor supports a flowchart that represents an operation algorithm. The expressions of the operation are in a form of various boxes and their order in a form of arrows. Fig. 2 presents the Odin model definition of a user-specific operation which returns all members that have a given birth year. The flowchart emulates the following steps of an algorithm:

- the declarations of the necessary local, input and output variables;
- the assignment of a list of all members to the appropriate variable;
- the assignment of the number that represents the size of the aforementioned list to the appropriate variable;
- the definition of a loop that executes the following steps while the value of the iterator is smaller than the size of the list number:

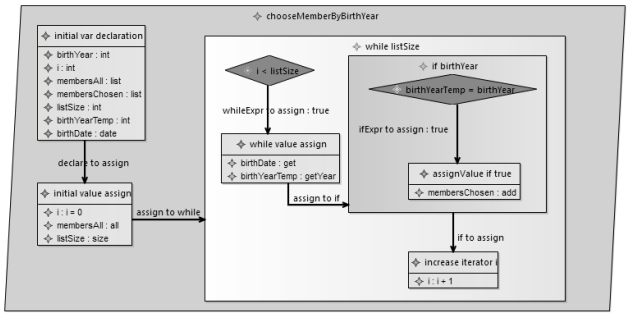

Figure 2. The user-specific operation for member selection

- isolate the birth date from a member that is on the iterator position in the list of all members;
- check if the year of the isolated date equals the given input year;
- add the member to the list of result members if the value of the check is *true*; and
- increase the iterator value by one.

Fig. 3 represents the XML listing of the example user-specific operation in the Odin model while Fig. 4 represents the Java listing of corresponding generated code.

```
<customMethods name="chooseMemberByBirthYear"
nameOfEntity="Member">
 <declareVariable name="initial var declaration">
  <variable varType="input" name="birthYear"/>  <variable name="i"/>
  <variable listOfEntity="Member" type="list" name="membersAll"/>
  <variable listOfEntity="Member" type="list" name="membersChosen"/>
  <variable name="listSize"/> <variable name="birthYearTemp"/>
  <variable type="date" name="birthDate"/>
  <declareToAssign
   declareToAssignTarget="//@customMethods.0/@assignValue.0"/>
 </declareVariable>
 <assignValue name="initial value assign">
  <value name="i" expression="i = 0"/>
  <value name="membersAll" listOp="all"/>
  <value name="listSize" listOp="size" nameOfList="membersAll"/>
  <assignToWhile assignToWhileTarget="//@customMethods.0/@while.0"/>
 </assignValue>
 <while name="while listSize">
  <whileExpr whileExpr="i &lt; listSize">
   <whileExprToAssign
    whileExprToAssignTarget=
    "//@customMethods.0/@while.0/@assignValue.0"/>
  </whileExpr>
  <assignValue name="while value assign">
   <value name="birthDate" dateOp="get" nameOfList="membersAll"
    posInList="i" fieldOfEntity="birthDate"/>
   <value name="birthYearTemp" dateOp="getYear"
    nameOfVariable="birthDate"/>
   <assignToIf
    assignToIfTarget="//@customMethods.0/@while.0/@iff.0"/>
  </assignValue>
  <assignValue name="increase iterator i">
   <value name="i" expression="i + 1"/>
  </assignValue>
  <iff name="if birthYear">
   <ifExpr ifExpr="birthYearTemp = birthYear">
    <ifExprToAssign
     ifExprToAssignTarget=
      "//@customMethods.0/@while.0/@iff.0/@assignValue.0"/>
   </ifExpr>
   <ifToAssign  ifToAssignTarget=
    "//@customMethods.0/@while.0/@assignValue.1"/>
    <assignValue name="assignValue if true">
     <value name="membersChosen" listOp="add" nameOfList="membersAll"
      posInList="i"/>
    </assignValue>
  </iff>
 </while>
</customMethods>
```

Figure 3. The XML listing of the user-specific operation

```
//left out code
private int birthYear;
public void chooseMemberByBirthYear() {
 try {
  List<Member> membersAll = new ArrayList<>();
  List<Member> membersChosen = new ArrayList<>();
  int i = 0;
  membersAll = adminMemberBeanRemote.listAllMembers();
  int listSize = membersAll.size();
  while (i < listSize){
   Calendar birthDate = Calendar.getInstance();
   birthDate.setTime(membersAll.get(i).detail.getBirthDate());
   int birthYearTemp = birthDate.get(Calendar.YEAR);
   if (birthYearTemp == birthYear){
    membersChosen.add(membersAll.get(i));
   }
   i++;
  }
  members = membersChosen;
//left out code
```

Figure 4. The Java listing of the user-specific operation

The portability of the generated code to another programming language requires nothing more than the definitions of the appropriate code generation facilities.

V. CONCLUSION

This paper addresses the problem of the user-specific operation support in a web business application development. The paper deals with a platform-independent specification of such operations by representing operation algorithm steps in a form of a customized flowchart. A flowchart is a popular diagram for an algorithm description. The proposed approach puts the development focus on an algorithm rather than implementation details of an underlying programming language. The approach allows developers to model user-specific operations only once and there is no need for new specifications across different platforms. By this, the same user-specific operations do not have different declarations for different platforms.

By adding support for user-specific operations to the OdinModel framework, we overcome the shortcomings of the existing model-driven and non-model-driven tools. To our knowledge, the visual modeling of user-specific operations and the full code generation from a single abstract model makes the proposed approach unique among model-driven tools. Support for an underlying data model provides sharing of data between operations, databases, and object-oriented models - something that the existing non-model-driven tools do not support. The portability of the generated code to another platform only requires the appropriate code generator definitions. The proposed approach extends the basic set of generic CRUD operations and fulfills the whole model-driven concept of one model as a formal and complete definition of an application.

For the future work, the planned task is to add the other algorithm steps, such as new data types, constants, for loop, and keywords. Another task is to provide a visual editor with semantic checks. Developing of a textual editor with both semantic and syntax checks is in the plan. One of the planned tasks is developing of code generation facilities for other development languages. Developing separate plug-in outside of OdinModel that would support external tools only with user-specific operations is in order.

ACKNOWLEDGMENT

Results presented in this paper are part of the research conducted within the Grant No. III-44010, Ministry of Education, Science and Technological Development of the Republic of Serbia.

REFERENCES

[1] Cuadrado, J.S., Izquierdo J.L.C., Molina J.G.: Applying model-driven engineering in small software enterprises. In: Science of Computer Programming 89. pp.176-198 (2014)

[2] Čalić, T., Dascalu S., Egbert D.: Tools for MDA software development: Evaluation criteria and set of desirable features. In: Information Technology: New Generations 5. pp. 44-50. IEEE (2008)

[3] Voelter, M., Benz S., Dietrich C., Engelmann B., Helander M., Kats L.C., Visser E., Wachsmuth G.: DSL engineering: Designing, implementing and using domain-specific languages. dslbook. org (2013)

[4] Balać, V., Vidaković, M. Extendable Multiplatform Approach to the Development of the Web Business Applications. In: Konjović, Z., Zdravković, M., Trajanović, M. (Eds.) ICIST 2016 Proceedings Vol.1, pp.22-27 (2016)

[5] Groenewegen, D.M., Hemel Z., Kats L., Visser E.: Webdsl: a domain-specific language for dynamic web applications. In: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. pp. 779-780. (2008)

[6] Popovic, Aleksandar, Ivan Lukovic, Vladimir Dimitrieski, and Verislav Djukic. "A DSL for modeling application-specific functionalities of business applications." *Computer Languages, Systems & Structures* 43 (2015): 69-95.

[7] Flowgorithm [Online] Available: http://www.flowgorithm.org/ (Current March 2017)

[8] Larp [Online] Available: http://www.marcolavoie.ca/larp/en/default.htm (Current March 2017)

[9] Raptor [Online] Available: http://raptor.martincarlisle.com/ (Current March 2017)

[10] Visual logic [Online] Available: http://www.visuallogic.org/ (Current March 2017)

[11] DRAKON Editor [Online] Available: http://drakon-editor.sourceforge.net/ (Current March 2017)

[12] Blockly [Online] Available: https://developers.google.com/blockly/ (Current March 2017)

[13] Cook, S.: Domain-specific modeling and model driven architecture. MDA Journal (2004)

[14] Selic, B.: The pragmatics of model-driven development. In: IEEE software 20, no. 5. pp.19-25 (2003)

[15] Sirius [Online] Available: https://eclipse.org/sirius/ (Current March 2017)

[16] Acceleo [Online] Available: https://eclipse.org/acceleo/ (Current March 2017)