Generate User Interface Using Xtext Framework

Dragana Aleksić[1], Dušan Savić[1], Siniša Vlajić[1], Alberto Rodrigues da Silva[2] Vojislav Stanojević[1], Ilija Antović[1], Miloš Milić[1],
Faculty of Organizational Sciences, University of Belgrade [1]
Department of Computer Science and Engineering,IST / University of Lisbon[2]

Abstract- A very important aspect of the software development process is the cost of completing the software project. The development of the user interface is a significant part of such a process. Typically, the graphical user interface of an interactive system represents about 48% of the source code, requires about 45% of the development time and 50% of the implementation time. Therefore, if this part of the software development process can be automated in some way, it can reduce the cost of a software project. In this paper we propose a small domain-specific language for specification of a graphical user interface and present a generator for Java desktop applications. The generator should facilitate the development of a software system by automatic creation of immutable code and creation of variable code based on parameters of the input specification of the generator. For this purpose we have used an Xtext open-source framework. A small but representative example is shown to describe the whole process.

Keywords: generator, prototype, user interface, domain specific language

I. INTRODUCTION

While creating software one comes across various problems. Many of them are related to collecting user requirements and overall time needed for designing and creating graphical user interface (GUI). User requirements are often represented by verbal description provided by end user or/and domain expert together with programmer. Discovered in later phase of lifecycle development of software system, incomplete and incorrect user requirements can increase cost and overall time needed for development of entire system.

In accordance with this a very important aspect in software development is the cost of completion of a software project. Development of the user interface is a significant part of such a process [1].Development of user interfaces (UIs), ranging from early requirements to software obsolescence, has become a time-consuming and costly process. Typically, the graphical user interface of an interactive system represents about 48% of the source code, requires about 45% of the development time and 50% of the implementation time, and covers 37% of the maintenance time [2].

A step forward in software industry towards solution of above problems is creation of various code generators which generate parts or even entire software systems of particular domain. Code generator is a component that generates appropriate software system from input specification model usually expressed in some modeling language. Modeling language as a set of all possible models that are conformant with the modeling language's abstract syntax, represented by one or more concrete syntaxes and that satisfy a given semantics [3]. The code that is generated by generator is divided into: 1) Generic codes that represents a code which is independent of concrete applications and can be used in other applications of the same type without changes, and 2) Specific codes that represents the code that is specific to a particular application or can be seen as some templates, upon which program code is created. In this paper we introduce a UIL domain specific language (DSL) for user interface specification implemented using Xtext framework, and appropriate generator. Xtext is based on openArchitectureWare generator framework, the Eclipse Modeling Framework and Another Tool for Language Recognition (ANTLR) parser generator. This paper is organized as follows. Section II describes the background of this work while Section III present related work. Section IV presents proposed DSL and appropriate code generator. Section V concludes the paper and outlines future work.

II. BACKGROUND

According to [4], domain specific language is a programming language of limited expressiveness focused on particular domain. The term “limited expressiveness” refers to the fact that DSLs only have minimal features needed to support its domain as opposed to general purpose languages (GPLs) which provide a lot of capabilities that can be applied on various domains.

Language workbenches were created after some time as supporting tool for creation of external DSLs. They are specialized environments which facilitate development of DSL by providing a good support for creation of both concrete and abstract syntaxes, automatic creation of underlying parser and complete editing environments for using DSLs with all the necessary support needed for creating a program like syntax highlighting, code

completion, customizable outline and real-time constraint checking.

Model Driven Development (MDD) looks at the software development lifecycle through modeling and transformation of one model into another. MDD's goal is to replace traditional methods of software development with automated methods with domain-specific languages which efficiently express domain concepts and directly represent domain problem. Such development directs traditional programming into creating domain-specific models and code generators which further leads into higher software productivity, therefore, increased satisfaction of the end user.

Code generators make software development process easier by shortening time necessary for writing programming code as well as time for testing the generated code, considering that generated parts of system was tested already. Generators can be used for creating parts of the system or for creating whole system as a prototype which can be rejected, upgraded or kept as a final solution.

III. RELATED WORK

Today there are so many modeling languages. Modeling languages might be classified as general purpose or domain-specific modeling language [5], [6] The one part of them are User-Interface Modeling Languages such as UMLi, UsiXML, WebML (with IFML), XIS, DiaMODL, XIS-Mobile and etc. Morais and Silva [7] introduce ARENA framework and evaluate the quality and effectiveness of modeling languages [8]. Ribeiro and Silva in their paper [9] present XIS-Mobile language which is defined as UML profile and supported by the Sparx Systems Enterprise Architect. XIS-Mobile is part of the broader XIS project that considers three major groups of views: Entities, Use-Cases and User-Interfaces. [10] [11]. XIS focuses on the design of interactive software systems at a Platform-Independent Level according to MDA. XIS-Mobile language reuses some concepts proposed on the XIS language and introduces new ones in order to be more appropriate to mobile applications design. It has a multi-view organization and supports two design approaches: the dummy approach and the smart approach. Dejanovic at al. [12] present an extensible domain specific language (DOMMLite) for static structure definition of database-oriented applications. Also, they developed appropriate textual Eclipse editor for DSL from which they generate complete source code for GUI forms with CRUDS operations. [13]. Popovic at al. [14] proposed IIS*CFuncLang DSL to enable a complete specification of application-specific functionalities at the PIM level and developed algorithms for transformation of IIS*CFuncLang specifications into PL/SQL program code. Furthermore, they presented tree-based and textual editors that are embedded into IIS* Case development tool. Tran at al. REF [15], [16] proposed a process that combines the task, domain, and user model in order to design user interface and generate source code. They propose framework as well as software prototype tools named as DB-USE. Kennard has emphasized the need for UI generation within mainstream software development and explains five characteristics, which he believes, are key to a practical UI generator. These characteristics are discovered through interviews, adoption studies and close collaboration with industry practitioners [17]. Cruz an Faria proposed an approach to create platform independent UI models for data intensive applications, by automatically generating an initial UI model from domain and use case models. [18], [19] defined extensions for domain model and use case model to support user interface generation. They describe how both of these models could be mapped in user interface features. Pastor at al. proposed full Model Driven Development approach from requirements engineering to automatic code generation by integrating two methods: Communication Analysis and OO-Method [20]. Smiałek and Straszak presented a tool that automates transition from precise use case and domain models to code [21]. Smiałek at el. defined RSL as a semiformal natural language that employs use case for specifying requirements [22]. Each scenario in a use case contains special controlled natural language SVO (O) sentence. RSL has been developed as a part of ReDSeeDS project [23]. ReDSeeDS approach covers a complete chain of model-driven development – from requirements to code [24]. Requirements is described in form of use cases using constraint requirements specification language which can be used to generate complete MVC/MVP code structure.

Inside our Laboratory we developed integrated SilabMMD approach [25] that use SilabReq language [26], [27] for specifying user requirement which is implemented inside JetBrains Meta Programming System and can be used as plug-in for IntelliJ IDEA or for the MPS tools. In [28] we proposed use cases specification at different levels of abstraction to promote better integration, communication and understanding among involved stakeholders. Using this approach different software development artifacts such as domain model, system operations, user interface design can be automatically generated or validated. In [29] we identified the correlations between the use case model,

data model and the desired user interface, and purpose different ways of user interaction with the system and recommended the set of most common user interface templates.

IV. DEVELOPMENT OF DSL AND USER INTEREFACE GENERATOR

Developed generator creates GUI based on model description which is user defined with UIL DSL. Depending on model description, developed generator can produce limited scope of different user interfaces.

Desired DSL should be able to describe domain model like any other modeling language (for instance UML) in addition with information how domain object and their attributes are represented in GUI. Based on detail model description, developed generator automatically generates executable code, providing user with functional GUI in a few seconds. If not satisfied with the look, user can easily change DSL script and quickly see how changes he or she made are reflected in the final design.

Creation of user interface generator can be divided into several phases. First phase represents development of domain specific language for describing input specification of generator. The second phase refers to designing the architecture of software which is the expected output from the generator. Third phase covers the analysis of reference code implementation during which are noticed general and specific parts of the system. And final, fourth phase represents defining code generator.

A. DEVELOPMENT OF DOMAIN SPECIFIC LANGUAGE

As [4] points out, development of external DSL is very similar to development of GPL in a way that both have key parts that makes programming language: abstract syntax, one or more concrete syntaxes and whole process of parsing input including lexical and syntactic analysis.

For each domain object, user chooses whether the object will have representation or not. Attribute type supported are `int`, `double`, `boolean`, `String`, `Date` and reference type. Based on type of attribute, UIL DSL provides several graphical implementation choices, from which user, while describing attribute of certain object, can choose. Graphical representations supported are `TextField`, `TextArea`, `NumberPicker`, `RadioButton`, `CheckBox` and `ComboBox` which is default representation for reference. For establishing relation between object UIL DSL provides two graphical components: table and list which are visually placed in separate window and populated with collection of appropriate objects Also, user is able to customize certain configuration of graphical component, like minimal or maximal value in `NumberPicker` component or add some validation like regular expressions to text based components. An extract of our UIL DSL grammar is represented below.

```
Model: 'Domain objects:'
         objects+=Object*;
Object:
'{'  (createForm?='create form' (','''number of columns:'
columns = INT ','''number of rows:' rows = INT)? ','')?
'name:'name = ID ','''attributes:'  attributes+=Attribute*
'}';
Attribute: (PrimitiveType | ReferenceObject);
ReferenceObject:
         '{'''name:'name=ID ':' type= [Object] '['minLimit=
Limit '..' maxLimit = Limit ']'
          (','''representation:' representation =
('ComboBox') (','''label name:' label= STRING)?)?
          (','''position:''['column = INT ';' row = INT']')?
          (','''selection:' selection = SelectionType)?
'}';
PrimitiveType: '{'''name:'name=ID ':' type=  Type
(','''position:''[' column = INT ';' row = INT ']')?
(',' required?='required')?
'}';
SelectionType: name = ('List' | 'Table');
Type: (TypeString | TypeInt | TypeBoolean | TypeDouble |
TypeDate) ;
TypeString:name ='String' (','''representation:'
representation = (TextArea |TextField))?;
TypeInt: name ='int' (','''representation:' representation =
(TextField | NumberPicker))?;
TypeDate: name ='Date' (','''representation:' representation
= (TextField | NumberPicker))?;
TypeBoolean: name ='boolean' (','''representation:'
representation = (RadioButton | CheckBox))?;
RepresentationType: TextField | TextArea | NumberPicker |
RadioButton |CheckBox;
TextField: name ='TextField' (','''label name:' label=
STRING)? (','''regex:' regex= STRING)?;
NumberPicker: name = 'NumberPicker' (','''label name:' label=
STRING)? (','''initial value: ' initial = STRING)? (','''min
value: ' min = (STRING /*|'null' */ ))?
(','''max value: ' max = (STRING /*| 'null' */))? (','''step:
' step = STRING)? (','''format: ' format = STRING)?;
RadioButton: name = 'RadioButton' (','''label name:' label=
STRING)? (','''option true:' optionTrue = STRING)?
(','''option false:' optionFalse = STRING)?;
```

Let's define domain object called Pet with attribute race as type String, for which we don't want graphical representation. Next, we define Person with attributes first name and last name as type String, attribute date of birth as type Date and relation towards Pet with cardinality zero – to – many since in our model person can have none or many pets. While defining attributes of object, we have to specify required graphical representation that is `textfield` for first and last name, `numberpicker` for date of birth and `combobox` for relation. Since Person has relation with Pet we have to specify desired selection, in this case we choose table. For all graphical representation we can specify accompanying text as decryption of attribute that will be seen on GUI as label. Further, for `numberpicker` we can specify initial, minimal and maximal value as the format in which date will be represented. Also, we have

an option to specify positions where concrete attribute will be placed inside of GUI by providing desired number of columns and rows and exact index of pair column/row for each attribute. We want all attributes of Person to be arranged in one column.

```
Domain objects:{
         name: Pet,
         attributes:
         {name: race: String} }
{create form,
name: Person, number of columns:1, number of rows:4,
attributes:
{name: firstName: String, label name:"First name",
representation:TextField,position: [1; 1],required }
{name: lastname: String, label name:"Last name",
representation:TextField,position: [1; 2]}
{name: dateOfBirth: Date,
representation:NumberPicker,
         label name:"Date of bith",
         initial value: "05.02.1900",
         step: 'Calendar.YEAR',
         format: "dd.MM.yyyy",
         position: [1; 3] }
{name: pets: Pet [0..*],
         representation:ComboBox,
         position: [1; 4],
         selection:Table}
}
```

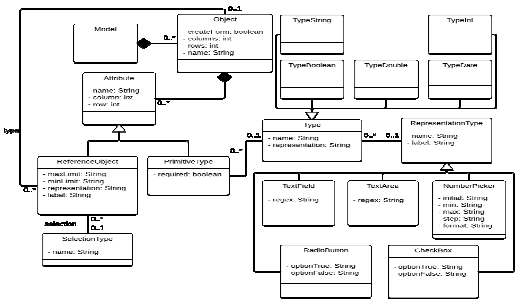

Figure 1 – Meta – model of UIL DSL

Abstract syntax describes concepts of programming language and their relationships. Fig.1 presents a meta – model of our UIL DSL. Concepts that are used in developed domain specific language are concepts that are tightly connected to the concepts of problem domain for which generator is developed. Given that the main concepts of UIL DSL are objects of domain model, their attributes, types and graphical representation of attributes. Concepts of abstract syntax can be represented with various concrete syntaxes. In Xtext, abstract syntax is represented in a form of ecore model and it is automatically created by Xtext based on defined concrete syntax.

Semantics defines the meaning for syntactically valid program by providing a set of rules which unambiguously specifies the meaning of a program. According to [30], semantics of language is translation of inputs into some target language which already has some behavior definition for its elements.

One way that DSLs usually differ from GPLs is that DSLs introduce one more intermediate layer called *semantic model* which is created after the parsing process is done and populated with data from abstract syntax tree. [4] points out that the semantic model defines semantic of DSL.. We implemented various validation rules, which will prevent end user from making semantically incorrect statements. Below is given one validation rule for cardinality between objects which prevents upper bound to be smaller than lower bound and also prevent upper bound to be zero.

```
@Check
publicvoid checkMinAndMaxLimit(ReferenceObject object) {
         if (object.getMaxLimit().equals("0")) {
error("Max limit cannot be 0",
FormDslPackage.Literals.REFERENCE_OBJECT__MAX_LIMIT);}
         if(object.getMinLimit().equals("*") &&
!object.getMaxLimit().equals("*")){
error("Min limit cannot be greater than max
limit",FormDslPackage.Literals.REFERENCE_OBJECT__MIN_LIMIT);
}}
```

After all changes in DSL script have been saved, generator begins with parsing process. Behind the scene, Xtext is integrated with ANTRL which generates parser for defined concrete syntax and population of abstract syntax tree with input data. Semantic model is automatically built from abstract syntax tree.

B. DESIGNING OF ARCHITECTURE OF SOFTWARE

After development of DSL, next phase is designing desired architecture of software system which will be generated. Software system needs to be well structured so it can be easily extended and maintained. We accomplished good structure of the system by implementing design patterns and SOLID principles.

C. ANALYSIS OF REFERENCE CODE IMPLEMENTATION

Third phase deals with creation and analysis of reference code implementation. Reference code implementation is done manually and serves as reference to what is the expected output from the generator. Reference software system should completely implement architecture designed in previous phase and should be extensive enough to cover all different use cases of generator and their implementations. After creation of reference software system, analysis is carried out during which specific and generic parts of system are diagnosed. Generic parts are ones that stay unchanged for different domains while specific parts are ones that change for particular domains.

D. DEFINING CODE GENERATION

Creation of code generator covers defining a transformation of concepts of DLS's meta – model into appropriate concepts of target platform – Java. Defining a

transformation means defining a set of rules of translation of inputs into outputs. For example we have to define translation between `NumberPicker` concept of UIL DSL and `JSpinner` component in Java. Given transformations, i.e. code generation is considered as translation semantics of programming language. Actual input in code generator is semantic model, created and populated with data (description of model which user defined) by XText after the parsing process is done. In other words, code generator is tightly coupled to semantic model since it reads all input data from it

For generic and specific parts diagnosed in the third phase, we look up for pattern they represent not related to any particular domain and upon that pattern we write transformations which will produce desired output. Code generation can be done in two ways, via transformer generation and template generation. In transformer generation we write programming code – transformation which represents logic of transforming input data into instructions of desired language. In template transformation we write templates which consist of static, generic parts that are same for all inputs and dynamic, specific parts that vary based on input. Specific parts, often called markers, retrieve data from semantic model and during compilation are replaced with real data. In our generator we use both approaches since they both have their advantages. Below is given a part of one template which generates table model for table component of UIL DSL. Table models consist of parts that vary based on different domain object, but there are still parts that are same for all, together those parts make a pattern – template upon which all table models will be generated.

```
public class
TableModel«UtilMethods::toFirstUpperCase(obj.name)» extends
TableModel { String columnNameFK[] = { "yes/no",
«UtilDomainAttribute::getAttributesName(obj)» };
public
TableModel«UtilMethods::toFirstUpperCase(obj.name)»(List<Wra
pperObject> lista) {
                this.columnName = columnNameFK;
                this.lista = lista;}
@Override
public Object getValueAt(int rowIndex, int columnIndex) {
        WrapperObject object = lista.get(rowIndex);
                switch (columnIndex) {
        «UtilDomainAttribute::getCasesForGetValueAt(obj)»
        }
        return "error";
}}
```

Based on output type of generator, transformation belongs to group M2C (model - to - code) since we transform input semantic model into Java source code. Generated source code is ready for execution without any modification, although user needs to populate a list of domain objects which will be used for selection if object has one – to – much relationship and wants to show that relation in GUI. Mixing generated and hand written code is not a good practice simply because all hand written changes will be overwritten next time generator runs. Given problem is solved by implementing Generation Gap pattern which with concept of inheritance decouples generated from hand written code. Fig.2 shows the results of code generation that are two forms, main form for inserting data about person and selection form for choosing objects of pets. User needs to populate decoupled list of pets with desired objects.

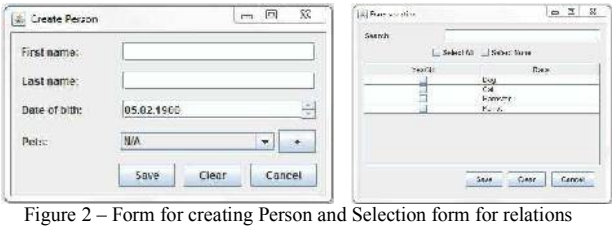

Figure 2 – Form for creating Person and Selection form for relations

IV. CONCLUSION

Fully developed generator eases the process of creating UI by freeing the user from direct implementation of programming code. Authors have acknowledged strengths of domain-specific languages, huge advantages of developing a code generator, in this case, GUI generator and great potential of MDD.

For creating a case study found in this paper, we have spent far less amount of time than for the development of similar projects using a traditional approach (general-purpose language). It can be concluded that the development of domain-specific language and code generator is definitely one of the ways to solve the problem of total amount of time necessary for full system software development and significantly reduce number of problems related to collecting of user's requests.

REFERENCES

[1] Kennard, R., Leaney, J.: 'Towards a general purpose architecture for UI generation', J. Syst. Softw., 2010, 83, (10), pp. 1896–1905

[2] Myers, B.; Rosson, M. B.: Survey on User Interface Programming. Proc. of the 10th Annual CHI Conference on Human Factors in Computing Systems, pp. 195-202, 2000.

[3] Alberto Rodrigues da Silva, Model-Driven Engineering: A Survey Supported by a Unified Conceptual Model, in Computer Languages, Systems and Structures, Elsevier, 43, 2015

[4] Martin Fowler, Domain-Specific Languages. The Addison-Wesley signature series, Addison-Wesley 2011, ISBN 978-0-321-71294-3, pp. I-XXVIII, 1-597

[5] Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. ACM Computing Surveys, 37:316–344.

[6] Luoma, J., Kelly, S., and Tolvanen, J.-P. (2004). Defining domain-specific modeling languages: Collected experiences. In OOPSLA 4th Workshop on Domain-Specific Modeling.ACM.

[7] Francisco Morais and Alberto Rodrigues da Silva, Assessing the Quality of User-Interface Modeling Languages

[8] André Ribeiro, Alberto Rodrigues da Silva, Evaluation of XIS-Mobile, a Domain Specific Language for Mobile Application Development

[9] André Ribeiro & Alberto Silva, XIS-Mobile A DSL for Mobile Applications

[10] Silva, A.R.: The XIS Approach and Principles, Proc. of the 29th Conf. on EUROMICRO (EUROMICRO '03), IEEEComputer Society (2003)

[11] Silva, A.R. et al.: XIS-UML Profile for eXtreme Modeling Interactive Systems, Proc. of the 4th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'07), IEEE Computer Society (2007)

[12] A Domain-Specific Language for Defining Static Structure of Database Applications

[13] Milosavljević, G., Perišić, B.: A method and a tool for rapid prototyping of large-scale business information systems, Computer Science and Information Systems ,vol. 1, pp. 57-82 (2004)

[14] A DSL for modeling application-specific functionalities of business applications

[15] Generating User Interface from Task, User and Domain Models

[16] Generating User Interface for Information Applications from Task, User and Domain Models with DB-USE

[17] Kennard, R., Leaney, J. Is there convergence in the field of UI generation, Journal of Systems and Software, 2011

[18] Cruz, A.M.R., Faria, J.P..Automatic generation of user interface models and prototypes from domain and use case models. In Proceedings of the 4th ICSOFT (ICSoft 2009) , vol. 1, pp 169-176, Sofia, Bulgaria, INSTICC Press, July 2009.]

[19] Cruz, A.M.R. Automatic generation of user interfaces from rigorous domain and use case models. PhD thesis F.E.U.P., University of Porto, Portugal, 2010.

[20] Óscar Pastor, Marcela Ruiz, Sergio España (2011) From requirements to code: a full model-driven development perspective; Keynote talk at ICSOFT 2011.

[21] Michał Smiałek, Tomasz Straszak, Facilitating transition from requirements to code with the ReDSeeDS tool

[22] M. Smiałek, J.Bojarski, W.Nowakowski and T. Straszak, “Scenario construction tool based on extended UML metamodel”. Lecture Notes in Computer Science, 3713:414–429, 2005.

[23] M.Smialek and T.Straszak, “Facilitating transition from requirements to code with the ReDSeeDS tool”. RE 2012: 321-322

[24] M.Smialek, W.Nowakowski, N. Jarzebowski, A Ambroziewicz,” From use cases and their relationships to code” MoDRE 2012: 9-18

[25] Dušan Savić, Siniša Vlajić, Saša Lazarević, Vojislav Stanojević, Ilija Antović, Miloš Milić, Alberto Rodrigues da Silva, SilabMDD - A Use Case Model Driven Approach

[26] D.Savic, I. Antovic, S. Vlajic, V. Stanojevic and M. Milic, “Language for Use Case Specification “, Conference Publications of 34th Annual IEEE Software Engineering Workshop, IEEE Computer Society, 2011, Pages: 19-26, ISSN : 1550-6215, ISBN: 978-1-4673-0245-6, Limerick, Ireland, 20-21 June 2011, DOI: 10.1109/SEW.2011.9

[27] D.Savić, A.Rodrigues da Silva, S.Vlajić, S.Lazarević, I.Antović, V. Stanojević, M.Milić, Preliminary experience using JetBrains MPS to implement a requirements specification language, in Proceedings of QUATIC’2014 Conference, 2014 , IEEE Computer Society

[28] Savić, A.R. Silva, S.Vlajić, S.D.Lazarević, V.Stanojevic, M.Milić, M.Milic, “Use Case Specification at Different Abstraction Level“, 8th International Conference on the Quality of Information and Communications Technology, Lisbon, Portugal, 03-06.09. 2012

[29] I.Antović, S.Vlajić, M.Milić,D.Savić and V.Stanojević, "Model and software tool for automatic generation of user interface based on use case and data model," Software, IET , vol.6, no.6, pp.559-573, Dec. 2012 doi: 10.1049/iet-sen.2011.0060D

[30] A Taxonomy of Model Transformation. Mens, Tom / Gorp, Pieter Van. Amsterdam: Elsevier Science Publishers , 2006, Electronic Notes in Theoretical Computer Science, T. 152. 1571-0661