TRƯỜNG ĐẠI HỌC SƯ PHẠM TP HỒ CHÍ MINH
TẠP CHÍ KHOA HỌC
KHOA HỌC TỰ NHIÊN VÀ CÔNG NGHỆ
ISSN: 1859-3100 Tập 14, Số 12 (2017): 66-79

HO CHI MINH CITY UNIVERSITY OF EDUCATION
JOURNAL OF SCIENCE
NATURAL SCIENCES AND TECHNOLOGY
Vol. 14, No. 12 (2017): 66-79

Email: tapchikhoahoc@hcmue.edu.vn; Website: http://tckh.hcmue.edu.vn

A FRAMEWORK FOR GENERATING GRAPHIC USER INTERFACE SOURCE CODE FROM UML CLASS DIAGRAM

Tran Anh Thi*, Vu Thanh Nguyen

Faculty of Software Engineering, University of Information Technology, Vietnam,

Received: 10/10/2017; Revised: 09/11/2017; Accepted: 04/12/2017

ABSTRACT

Producing source code that implements the GUI takes a great deal of effort in software development, especially for interactive software systems. This work load, generally considered tedious and burdensome, is inadequately automated given the richness of conceptual design and behavior models generated in earlier stages of the development process. A few frameworks have been proposed for generating GUI code based on formal specification or code annotation, requiring extra work to be done in addition to conceptually designing the software system in question. We propose a mechanism that generates GUI code from UML class diagrams expressed in XMI. Our approach takes into account the associations between design concepts and their composition hierarchy that is explicitly expressed in the UML language.

Keywords: code generate, software abstraction, UML, XMI, graphic user interface.

TÓM TẮT

Một công cụ phát sinh mã giao diện người dùng từ lược đồ lớp trong UML

Xây dựng mã nguồn giao diện cho người dùng được xem là một công việc tốn kém cho những nhà phát triển phần mềm, đặc biệt là những phần mềm có độ tương tác cao. Những công việc này thường tẻ nhạt, tốn kém thời gian và trùng lắp. Đây cũng là công việc khó khăn trong giai đoạn đầu của thiết kế phần mềm khi các yêu cầu cũng như các mô hình ở mức khái niệm còn chưa rõ ràng. Hiện nay cũng có một số công cụ đưa ra hướng phát sinh mã giao diện tự động từ các đặc tả cũng như những chú thích từ mô hình. Hướng này đòi hỏi phải xử lí nhiều thông tin thêm vào cho mô hình đặc tả mà nó nằm ngoài khái niệm thiết kế hệ thống phần mềm. Chúng tôi đề xuất một hướng tiếp cận phát sinh mã giao diện người dùng từ mô hình lớp trong UML thông qua XMI. Cách tiếp cận này, chúng tôi dựa vào các tính chất của mối quan giữa các khái niệm trong thiết kế và hệ thống phân cấp rõ ràng trong UML nhằm cung cấp thêm thông tin cho bài toán phát sinh mã giao diện.

Từ khóa: phát sinh mã, trừu tượng hóa phần mềm, ngôn ngữ Mô hình hóa thống nhất (UML), XMI, giao diện người dùng.

1. Introduction

Software engineering has long been related to tools, theo- ries, and methods. It connected together for cost-effective software development [1]. However, connecting theories and tools for developing software simply do not exist. The consensus in this regard is that software engineers should explicitly consider the domain of the software system to be built. A domain is characterized by the business processes being automated or

* Email: *thitta.10@grad.uit.edu.vn*

the real world problem being addressed by a software program. Having a good understanding of software domains is essential for the success of any software development project [2]. Narrowing down the software domain will open the door for tailored methods and tools that enable a few activities in the development process to be automated. Consequently, it will effectively cut down the developmet cost and time. This is the rationale behind domain-specific modeling. There are many domains in software engineering such as mobile applications [3], robot applications [4], web applications [5], etc. In the desktop application based domain, the challenge is to combine the graphic user interface with the objects in conceptual model (UML class, entity relationship, etc,.). This means code generation in a desktop application from a model is a real challenge in the future.

Unfortunately, not much effort has been put in building framework for desktop applications from conceptual model. Our research in this direction has initially come to the idea of combining constraint and the UML (Unified Modeling Language) class in application as a domain-specific modeling framework for desktop applications. Producing source code that implements the GUI (Graphics User Interface) takes a great deal of effort in software development, especially for interactive software systems. This work load, generally considered tedious and burdensome, is inadequately automated given the richness of conceptual design and behavior models generated in earlier stages of the development process. A few frameworks have been proposed for generating GUI code based on formal specification or code annotation, requiring extra work to be done in addition to conceptually designing the software system in question. We propose a mechanism that generates GUI code from UML class diagrams expressed in XMI (XML Metadata Interchange). Our approach takes into account the associations between design concepts and their composition hierarchy that is explicitly expressed in the UML language.

The remainder of this paper is organized as follows. *Section II* gives the preliminaries of our work. *Section III* presents our research motivation and formulates our research problems. Section IV is the core of our paper – a framework that explicitly addresses the research problems identified. *Section V* reports experiments conducted on our framework. *Section VI* surveys related work. *Section VII* draws some concluding remarks and points out the future work.

2. Background

2.1. The Model Driven Architecture

The Model Driven Architecture (MDA) is a framework for software development defined by the Object Management Group (OMG). Key to MDA is the importance of models in the software development process. Within MDA the software development

process is driven by the activity of modeling your software system [6]. The MDA development life cycle is not very different from the traditional life cycle. The artifacts of the MDA are formal models, i.e., models that can be understood by computers. The following three models are at the core of the MDA: Platform Independent Model (PIM), a model with a high level of abstraction that is independent of any implementation technology; Platform Specific Model (PSM), a model tailored to specify your system in terms of the implementation constructs that are available in one specific implementation technology. A PIM is transformed into one or more PSMs; Code, a description (specification) of the system in source code. Each PSM is transformed into code.

2.2. UML

The Unified Modeling Language[1] provides a variety of diagram type for integrated specification of both the structure and the behavior of system [7]. Currently, there are many tool to support development of software. It often do not only support the analysis and design of system, but also contain code generator to automatically. Basing XMI on the XML metalanguage by W3C.XMI (XML Metadata Interchange) intends to provide a standard way for users to exchange any kind of metadata that can be expressed using the MOF (Meta-Object Facility) specification by the Object Manage- ment Group (OMG) [8]. It integrates three industry standards: MOF (OMG), UML (OMG), and XML (W3C) [7].

2.3. XML and XMI

XML (eXtensible Markup Language) was envisages as a language for defining document formats for the Web as HTML [9]. XMI defines four elements (we refer to them as differential elements) used to support differential description of UML models: *XMI.difference* is the parent element used to describe differences from the base (initial) model; it may contain zero or more differences, expressed through the ele- ments *XMI.difference*, *XMI.delete*, *XMI.add* and *XMI.replace*. *XMI.delete* represents a deletion from the base model, *XMI.add* represents an addition to the base model and *XMI.replace* represents a replacement of a model construct with another model construct in a base model [10].

XMI document that represents the actual UML specification of this model. Each component change has a XMI-change document that specifies how a model version was constructed from the predecessor schema. As introduced before not only the UML models will be specified according to the XMI standard, but also model changes [11].

[1] http://www.uml.org

3. Motivation and research problems

3.1. Example

In this subsection, we briefly describe an example in which a developer needs to build a shopping application. Figure 1 is a UML class diagram for the conceptual design of this software management timetable in shopping manager. There are many classes, cardinality and relationships between of them. The *Customer* class is presented the customer and product is presented the product in shopping etc,. In this UML class diagram, there are some classes and relationship of every class. The *Customer* class corresponds to the customer in shopping. The *Order* class corresponds to the customer's order in shopping, etc. These classes in this diagram have many type relationships. For example, the *Account* class have aggregation relationship with the *Order* class. There are also many other relationships such as: Association, multiplicity, aggregation, composition, inheritance, generalization. In every class, there are many attributes. For example, in Customer class has *custome id*, *phone*, *address*, *email*, *node* attributes. We want to build a shopping application by this UML class diagram.

This application has some functions. It helps employees to make order, to find a product or to do something for reporting, progress and backup database. It must have a friendly interface running on the desktop. In addition, defining rules for mapping data types of class attributes to elements in the Java GUI is also a challenge. How to generate source code automatically for graphic user interface in this application?

3.2. Research Problems

The gap from such a high-level description of a shopping application given in *Subsection III-A* to concrete source code (e.g., in Java) poses a few research questions as follows.

• Leveraging a conceptual design to generate GUI widgets that correspond to all attributes declared for UML class diagram.

• Making GUI tables based on the multiplicity of the UML classes diagram specified in the conceptual design.

• Dealing with the hierarchy of the UML composition and the isomorphism of UML generalization in the conceptual design.

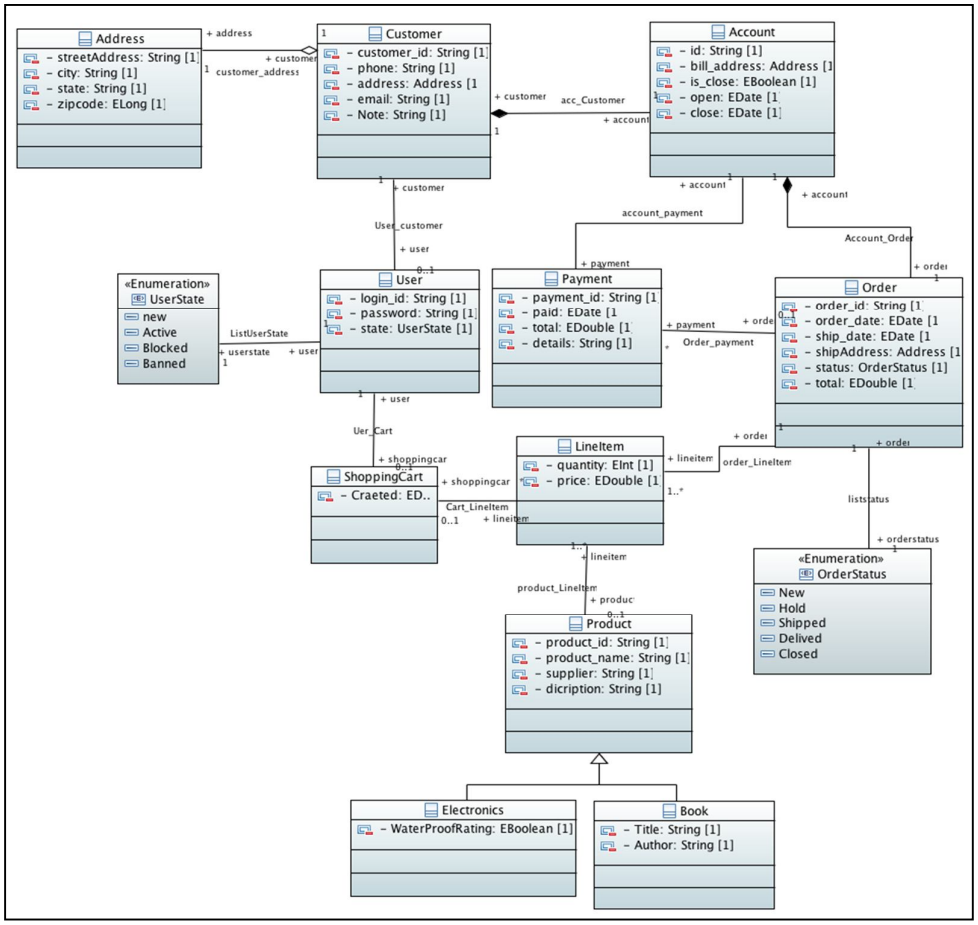


Fig. 1. *UML class diagram for shopping application*

4. Framework

In this section, we concretely present a proposed a framework with regards to software abstraction (*Subsection IV-A*), code generation and handling the dynamics of objects manipulated by shopping applications corresponding to the class diagram in the software abstraction (*Subsection IV-B*). Regarding the shopping application system depicted in *Figure* 2, there are many frames including single class, aggregation class, association class.

4.1. Software Abstraction

One of the pillars of a domain-specific framework is to raise the level of abstraction in making software specification. As we narrow down the domain (of software) to UML class diagram, we are able to employ a high-level declarative language such as the graphic. The classes are the objects in the shopping applications. The *Figure 2* is progress in our proposed framework that is described in *Figure 3*. Our framework has 4 steps to generate source code from UML class diagram.

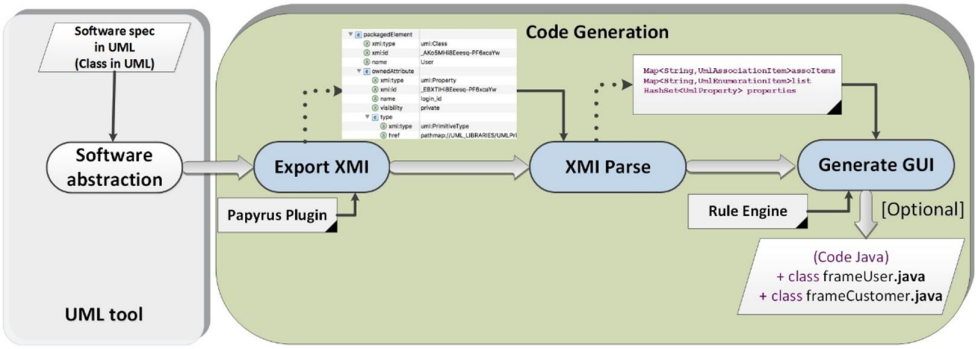


Fig. 2. *Our framework supports code generation of the graphics user interface from UML class diagram*

❖ *Step 1:* Creating a UML class diagram and defining attributes. This step, the software developers will design the class diagram for applications by any tool.

❖ *Step 2:* Mapping UML to XMI. By default, XMI is generated automatically for modeled classes. This step, we used Papyrus plug-in to parse the class UML.

❖ *Step 3:* Defining context. Show all objects which pro- gramer selects the object for the conceptual design. Algorithm 1 presents the main idea of this step. This algorithm parser the XMI document to list of classes, list of data- types, list of enumerations and list of associations. Result of *Algorithm 1* is to built a structure of the application. The H*CodeStruct* at line 6 corresponds to the structure of the application.

❖ *Step 4:* Generating GUI source code. *Algorithm 2* present the main idea of this step. This algorithm generates source code for the GUI in Java from structure of the application above and list of templates. In this step, the rule in *Table I* and *Table II* will be the basis for mapping attributes, relationships in UML class diagram with GUI in Java

4.2. Generating Source Code for Graphic User Interface

We also developed a simple tool that allows a software developer can generate code java for GUI widgets by class diagram [12]. They are declared class diagram by UML tool. Then, the engine will check classes and relation between classes, converted into XMI. In step 4, when the software developer has it taken steps in our framework, the program will automatically generate source code for the application. In which the structure of the application will be divided into a packages. View package will focus on the part of the graphic user interface, the package control will focus on the control handle and package model rule would specify the character in the application.

4.3. Attribute

The UML classes show the structural and behavioral fea- tures in the object oriented Model. These features include attributes, association, generation [7]. On the other hand, each attribute in a class diagram that contains information mapping to GUI. Thus, mapping

UML classes to Graphic User Interface are quite straightforward. In general, the UML attribute name is directly used as the GUI component name in the mapping process. However, class attributes in UML will have data types. In Table I, we built a rule that allowed us to map data types from UML to XMI, and then to the GUI component in Java.

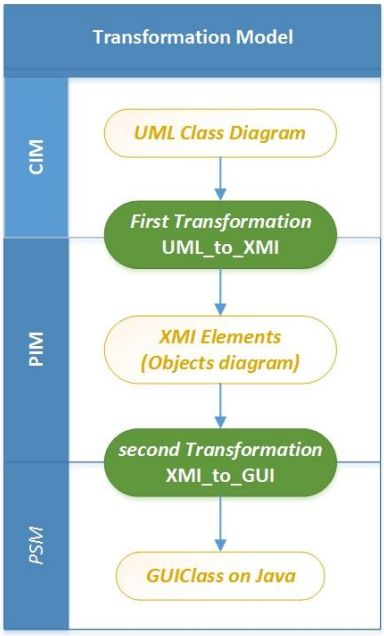


Fig. 3. *Transformation from UML class diagram to presenta- tion model in MDA*

4.4. Multiplicity

The problem is solving polymorphism. We take care of relationship of the UML class diagram generation. The GUI elements generated depends on attributes and demonstrate the relationship between classes. When programmer changes profile map layers, the GUI generating code section in the paper. This solution will update the GUI. Input parameters for the parser is the node iterator documentation of XMI. It is generated by using the class diagram papyrus tools. Element is a list of self-definition layer elements in XMI [13]. The build data structures to store information material was analyzed from XMI.

❖ A list of classes parsed from the XMI. Each class has a list of attributes and optionally methods.

❖ A list of UML relationships, which could be UML association or generalization. Each relationship links a pair of classes.

❖ A list of enumerations, each of which defines the data type for some attributes declared above.

Table 1. *Mapping Uml Class Diagram to Swing in Java for The Attributes*

UML Type	Java Type	Java Swing Object
EDate	Java.util.Date	JTextField
EByte, EShort, EInt	Interger	JTextField
ELong	Long	JTextField
EFloat	Float	JTextField
EDouble	Double	JTextField
EBoolean	Boolean	JRadioButton
String	String	JTextField
Object Reference	Reference Type	JCombobox

Algorithm 1: Parsing the XMI document to list of data-types, list of classes, list of enumerations and list of associations.

Input : UML File, XMI:Type $type$
Output: $H_{CodeStruct}$ the HashTable is defined by M_{Type}

$L_{XMLTypeElement} \leftarrow XMIParsing$(XML File);
foreach $t \in type$ **do**
  $M_{type}\ m \leftarrow XMIParsing.parseType(L_{XMLTypeElement}, t)$;
  $M_{Type} \leftarrow M_{Type} \cup m$;
end
$H_{CodeStruct} \leftarrow$ HashTable($keys$, M_{Type}); ▷ $keys$ = {Enum, DataType, Class, Association}
return $H_{CodeStruct}$;

Algorithm 2: Generate source code for the GUI in Java from list of classes, list of enumerations, list of associations and list of templates.

Input : $H_{CodeStruct}$ that is a list of data-type, classes, enumerations and associations from XMI document
$L_{TemplateSingle}$, $L_{TemplateDetail}$ those are the list of templates of graphic user interface defined
Output: $L_{PanelJava}$ that is a list of classes for graphic user interface in Java

foreach $h \in H_{CodeStruct}$ **do**
  switch $h.getkey()$ **do**
    case *Enumeration* **do**
      foreach $e \in h$ **do**
        generatedCodeSupport(e);
      end
    end
    case *DataType* **do**
      foreach $e \in h$ **do**
        generatedCodeSupport(e);
      end
    end
    case *Class* **do**
      foreach $e \in h$ **do**
        $L_{PanelJava} \leftarrow L_{PanelJava} \cup$ generatedCodeGUI($L_{TemplateSingle}$, e);
      end
    end
    case *Association* **do**
      foreach $e \in h$ **do**
        $L_{PanelJava} \leftarrow L_{PanelJava} \cup$ generatedCodeGUI($L_{TemplateDetail}$, e);
      end
    end
  end
end
return $L_{PanelJava}$;

In *Table 2,* we built a rule that allowed us to map relationships from UML to XMI, and then to the GUI component in Java. Every UML object responds to class in Java and object in Java Swing package.

Table 2. *Mapping UML class diagram to Swing in Java for the associations*

UMLObject	Java Code	Java Swing Object
Enumeration	JEnum	JCombobox
DataType	Support class	JCombobox
Class	Support class	Detail Panel
Association Multicipiliti 1..*	Support class	JTree[1]-DetailPanel[*]

5. Experiments

In this section, we report measurement we conducted for the application of our framework to two different applications. The first application is the running example described in *Subsection III-A. Table III* summarizes the UML class diagram for the specification and Java code generated by our code generation techniques for the implementation of the application. The sec- ond application is an application management. The *Figure 5* is the UML class diagram of this application. There are 5 classes and one interface. This application supports the management of employees in a company. Employees are classified into categories such as: Volunteers, executives and hourly. Each employee has common attributes and unique attributes. In the UML class diagram, there are some relationships between the classes, such as: Inheritance, generalization, realization.

Table 3. *Measurements conducted on the application of the code generation in our framework to 2 separate case-studies. As for the software abstraction, we report the number of class, association and inheritance of UML class diagram. As for Java source code generated, we report the number of Java classes, number of associations and inheritances captured in Java code and of course the number of Java line of code for each application.*

UML class diagram	Short description	Visualization	#Classes	#Relationships	#JFrames	Java LOC
Figure 1	The Shopping Application	Figure 4	**13**	**14**	**9**	**1,605**
Figure 5	The Management Application	Figure 6	**6**	**5**	**4**	**595**

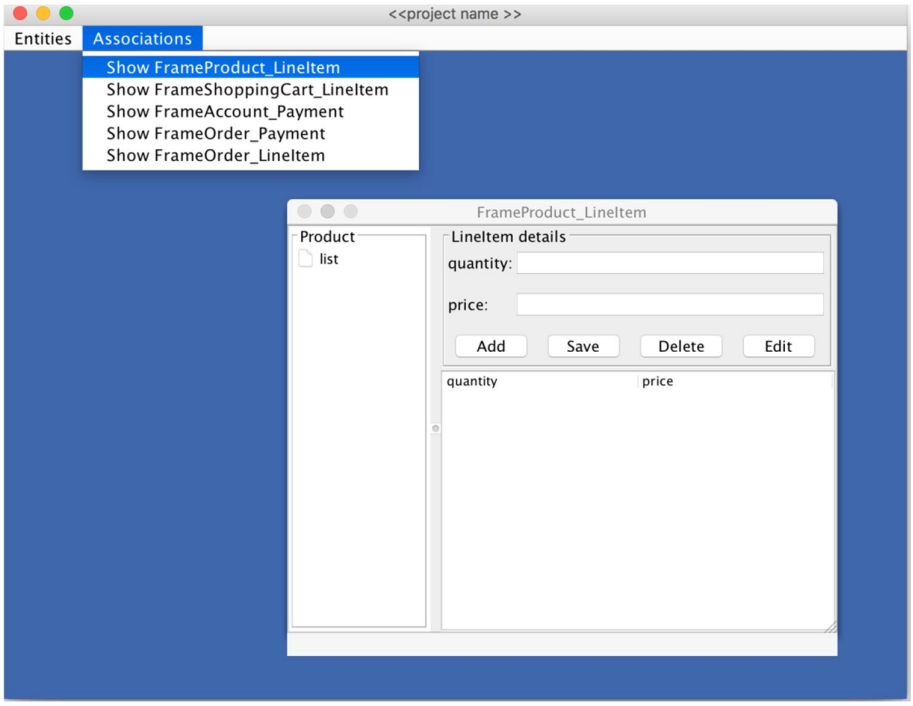
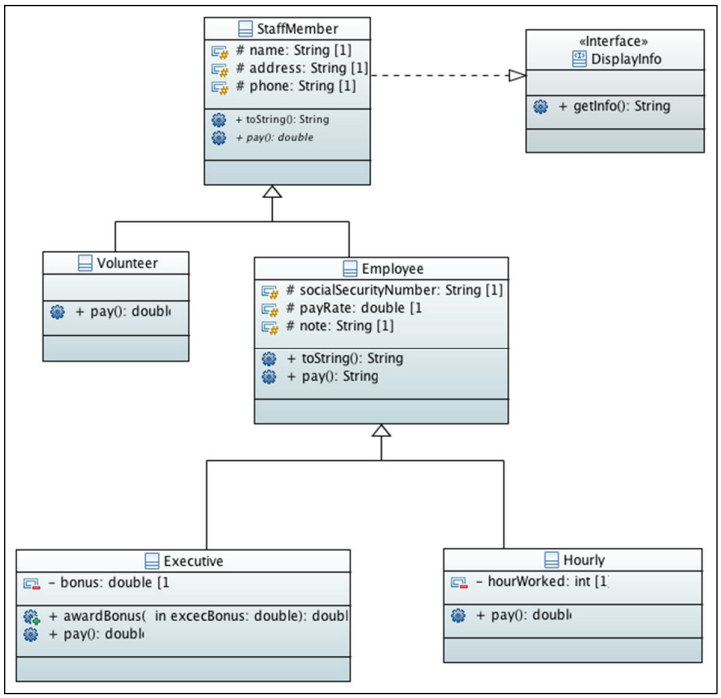


Fig. 4. *The GUI of shopping application is generated from our framework with UML class diagram*



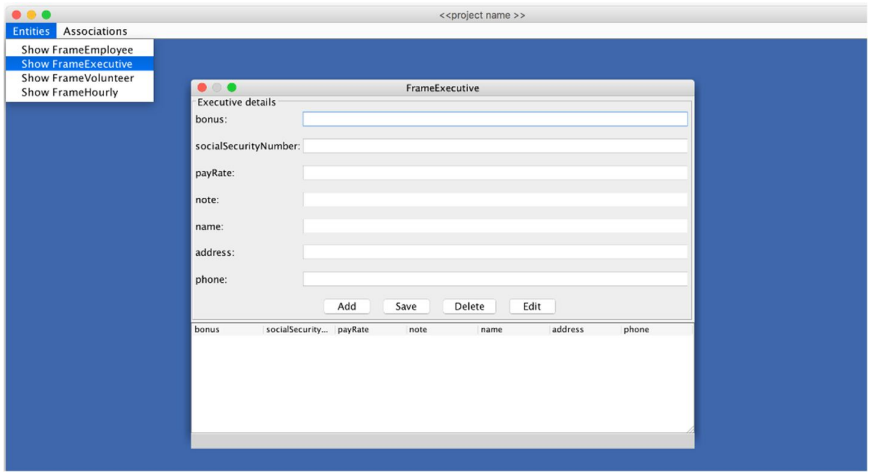


Fig. 6. *The GUI of management application is generated from our framework with UML class diagram*

6. Related work

6.1. MIDAS framework

MIDAS is a methodological framework for the agile devel- opment of WIS, which proposes a Model Driven Architecture based on the MDA approach. MIDAS proposes to model the WIS according to two orthogonal dimensions. On the one hand, taking into account the platform dependence degree (based on the MDA approach), two group of activities are considered: to specify the whole system by Computation Independent Models (CIMs), Platform Independent Models (PIMs) and Platform Specific Models (PSMs); and to specify the mapping rules between these models. On the other hand, MIDAS considers the modeling of the WIS according to three basic aspects: hypertext, content and behavior. Besides, MIDAS suggests using the Unified Model Language as unique notation to model both PIMs and PSMs [14]

6.2. UML and OCL

The Unified Modeling Language is popular specification. It allows to design structure, behavior, data structures and business processes model [8]. UML also uses diagrams to describe the models. A UML diagram does not provide con- straints between objects in models. However, every element has some constraints between of them. Those constraints must be declared by another language. That is The Object Constraint Language (OCL). The OCL is the constraint lan- guage of UML. It is a precise, declarative language that is simple to understand for people who are not mathematicians or computer scientists. It does not use mathematical operators, but maintains mathematical rigor in its definition [15]. The graphical notation of UML has no equivalent in textual style. Therefore, only with OCL is possible to specify additional constraints of the model in text. OCL can be used to specify

restrictions such as invariants, preconditions, post-conditions, among others. OCL is often referred as a "side-effects-free" language since the state of the system does not change due to an OCL expression.

6.3. Eclipse Modeling Framework

In the world of model-driven software development the Eclipse Modeling Framework (EMF) [16] is becoming a key reference. It is a framework for describing class models and generating Java code which supports to create, modify, store, and load instances of the model. Moreover, it provides gener- ators to support the editing of EMF models. EMF unifies three important technologies: Java, XML, and UML. Regardless of which one is used to define a model, an EMF model can be considered as the common representation that subsumes the others. For example, defining a transformation approach for EMF, it will become also applicable to the other technologies.

However, EMF have limited usability for code generated due to the following reasons: Limited capacities in construc- tion of visual representations of language concepts; Complex integration of different meta-models (DSLs); Lack of flexibil- ity in model transformations to an suitable target language; and Unsuitability for a specification of a larger amount of model variations [17].

7. Conclusions and future work

Software abstraction is welcome in the early phases of software engineering where model-based representations are to be shared by stakeholders whose interests differ substantially. In this paper, investigate how to make software abstraction tak- ing the viewpoints of requirements engineering and software design combined. We narrow down the scope of our work to desktop application. We presented how UML can be used as a domain modeling language. Our choice of UML class diagram as a model for software abstraction was justified in its ability to: (i) conceptually and declaratively specify class rules and application concepts and (ii) turn the formal representation of class rules into computer-interpretable models, which opens the door for further automation in the later phases of software development. In our paper, we have designed and implemented two algorithms (*Algorithm 1*, *Algorithm 2*) to mapping UML class diagram to components in Swing package of Java.

As for the future work, we consider the following research directions. First, we will to expand the scope of desktop applications (e.g., generate database) for making software abstraction. Second, we leverage the underlying engine of rule in generating source code for the implementation of the desktop applications. Third, we will develop mechanisms for monitoring the data structure of an application against its class rules.

❖ ***Conflict of Interest:*** *Authors have no conflict of interest to declare.*

REFERENCES

[1] I. Sommerville, *Software Engineering*. 10th Ed. New York: Pearson, April 2015.

[2] M. Fowler, *Domain-Specific Languages.* Pearson Education, 2010.

[3] D. Kramer, T. Clark, and S. Oussena, "MobDSL: A Domain Specific Language for multiple mobile platform deployment," in *Proceeding of the 2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications*. Suzhou, China, IEEE Computer Society, November 2010, pp. 1–7.

[4] A. Nordmann, A. Tuleu, and S. Wrede, "A Domain-Specific Language and Simulation Architecture for the Oncilla Robot," in *Proceedings of the ICRA 2013 Workshop on Developments of Simulation Tools for Robotics & Biomechanic*. Karlsruhe, Germany: IEEE Computer Society, May 2013, pp. 25–26.

[5] A. Charland And B. Leroux, "Mobile Application Development: Web vs. Native," *Communications Of The ACM*, Vol. 54, No. 5, pp. 49–53, 2011.

[6] W. B. A. Kleppe, J. Warmer, *Mda Explained: The Model Driven Architecture: Practice And Promise*. Boston, USA, Addsion Walley, 2003.

[7] A. Bruckerandj. Doser, "Metamodel-Based UML notations for Domain-Specific Languages," vol. 2, Zurich, Switzerland, 2007, pp. 25–26.

[8] D. Jackson, "A comparison of Object Modelling Notations: Alloy, UML and Z," *MIT, Tech. Rep.*, August 1999.

[9] E. Visser, "WebDSL: A case study in domain-specific language engineering," Springer Berlin Heidelberg, vol. 2, 2008.

[10] J. Kovse and T. Harder, "Generic XMI-based UML model transformations," *Object-Oriented Information Systems*, pp. 183–190, 2002.

[11] F. Keienburg and A. Rausch, "Using XML/XMI for tool supported evolution of UML models," in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. Hawaii, IEEE Computer Society, pp. 10–pp, 2001.

[12] C. Atkinson and T. Khne., "Concepts for comparing modeling tool architectures," *Springer Berlin Heidelberg*, 2005.

[13] R. Gronback, "Eclipse modeling project: a Domain-Specific Language (DSL) toolkit," *Addsion Walley*, vol. 20, 2009.

[14] J. Vara, V. DeCastro and E.Marcos,"WSDL automatic generation from UML models in a MDA framework," in *Proceedings of the International Conference on Next Generation Web Services Practices, 2005,* Seoul, South Korea, IEEE Computer Society, 2005, pp. 6–pp.

[15] D. Jackson, "Software Abstractions: logic, language, and analysis," *MIT press*, vol. 2, 2012.

[16] E. Biermann, K. Ehrig, C. Ko¨hler, G. Kuhns, G.Taentzer and E. Weiss, "Graphical definition of in-place transformations in the Eclipse modeling framework," in *Proceedings of the 9th IEEE/ACM International Conference on Model*, vol. 4199 of Lecture Notes in Computer Science. Springer, 2006, pp. 425–439.

[17] V. Djukic´, I. Lukovic´, A. Popovic´, and V. Dimitrieski, "Domain- Specific Modeling Tools as Client Applications Providing the Production of Documents," in *Proceedings of the Industrial Track of Software Language Engineering workshop*. Dresden, Germany: Springer, 2012, pp. 6–pp