A model-driven approach to automatically generating CRUD functionalities from use case stories: A Pet clinic example study

Nassima Yamouni Khelifi, Latifa Dekhici, Mahmoud Zennaki

SIMPA Laboratory (Signal, IMage, PArole)
Department of Computer Science
Faculty of Mathematics and Computer Science
University of Sciences and Technology of Oran -Mohamed Boudiaf-
nassima.yamounikhelifi@univ-usto.dz,latifa.dekhici@univ-usto.dz, mahmoud.zennaki@univ-usto.dz

Abstract--*One of the most critical challenges in model-driven research area is the automatic generation of the well-known CRUD (Create/Read/Update/Delete) functionalities form models. This work exploits the Model-driven Requirements Engineering (MDRE) approach that relies on requirements modelling to automate the production of a software application. An UML-like language is employed to model the initial user requirements. Moreover, this language is specific, and suggests constrained natural language case scenarios for more consistency, and coherence. The applicated translational rules take the input requirements models described as use case models accompanied with their domain models, and generates the CRUD implementation automatically from them. The resulted code adheres the well-known Model-View-Controller/Presenter (MVC/MVP) design pattern. However, it affects the Model layer, that is responsible on persisting and storing the data in a database system. The feasibility of the proposed method is shown through the pet-clinic example study. Even though, the presented transformation rules can be exploited for any platform, and any problem domain.*

Index terms*: model-driven, requirements modelling, CRUD, use case scenarios, MDE/MDD, automatic code generation, pet-clinic.*

I. Introduction and related works

Model-driven approaches (MDE/MDD/MDA, …) have been largely adopted as software development paradigms over the last two decades. They are based on three pillars: higher-level of abstraction (i.e.: use of models), higher-level of automation, and higher-level of standardization [1]. Furthermore, they have demonstrated the power of models in practically almost domains such as: mobile computing [2], cloud-computing [3], [4], cyber-physical systems [5], for embedded systems [6], geography [7], transport [8], [9] , e-health [10] , Internet of Things (IoT) [11], [12], … etc. The requirements are the initial entities in the software development process, and are meant to be clear and understandable for both the customers and the other stockholders such as the: software development environment**.** The generation of the software artefacts (models and code) is performed by two activities: Model-to-Model transformation (M2M), and Model-to-Text (M2T) transformations. The portion of the generated code is tightly dependent to the maturity of code generator (partial or full) [13].
The Model-driven Requirements Engineering (MDRE) discipline marries Requirements Engineering (RE) and MDE approaches to define the requirements with the help of models. These requirements have to be correct, consistent, unambiguous and easy to read/maintain [14]. The goal is to cope with the increasing complexity of technical systems [15] and to tackle the problem of automatic deliverance of a software application. Therefore, MDRE offers real benefits in term of productivity and quality [16], [17].
Several relevant works have been proposed to convert input models down to source code taking into account the support of the CRUD functionalities. Some of these approaches coincides with our approach in many points such as: model-based, level of automation of CRUD implementation, … and so on. Nevertheless, they differ in other points like: the source model (i.e.: requirements models or design models (PIM or PSM, …)), languages/tools: source language, translational language, target language, and their underlying platforms, … etc.
The work presented in [18] aims to automate the generation of the CRUD methods from data entities for data-intensive web applications, relying on Model-driven web engineering (MDWE). They have applied a pattern-based development approach for IFML (Interaction Flow Modelling Language) OMG standard. The AutoCRUD tool was developed on the top of WebRatio platform to reduce the specification of recurrent CRUD operations.
Very recently, the article of [19] discusses a model-driven method that combines low-code development and requirements engineering to semi-automatically produce a software business application. They transform requirements specified with the Application Specification Language: ITLingo ASL language into the low-code platform: Quidgest Genio. The generated application covers the CRUD operations.

The contribution of Hamza Ed-douibi et al. in [20] exploits a model-driven approach to generate (semi-) automatically OData services relying on relational databases from UML models. Within their approach, the generation of CRUD operations is fully supported.
In their proposal, Daniel Sanchez et al. [21] have used a method based on MDA principals to produce android applications from PIM model. The outputted Kotlin code supports the CRUD features.
The authors in [22] have introduced a framework that deals with the MDA methodology so-called: E-MDAV (Extended MDA-View), in which they automated the generation of CRUD functionalities and thus, gaining in time and effort of development. Their approach addressed data-intensive web-based applications.
The work presented in [23] allows to generate REpresentational State Transfer (REST) web services from textual requirements and domain models (CIM models) following the MDD/MDA paradigm: MDD4REST (semi)-automatically. Their approach considers the generation of CRUD, and non-CRUD operations with SQL and NoSQL databases too.
Christoforos Zolotas et al. in [24] proposed a model-driven architecture variant to convert textual requirements and visual story boards to an operational RESTful web service including crud and non-crud functionalities. The same authors have introduced the RESTsec low-code mechanism in [25], to offer secure enterprise services rapidly and automatically. They generate the standard CRUD operations, in addition to ABAC access control code to provide authorization capabilities for each CRUD operation.
The approach of [26] aimed to produce an MVC2 web application based on MDA technology from high-level CIM models. The web-based application code comprehends the CRUD operations.
Amine Moutaouakkil and Samir Mbarki in [27] have provided a method for transforming Struts model into CodeIgniter model following the MDA discipline. Their code encapsulates the basic CRUD methods.
Herein, the presented study is based on Model-driven Requirements Engineering principals to project the automatic generation the CRUD implementation from use case scenarios with their visual domains, according to the Model-View-Controller/Model-View-Presenter (MVC/MVP) architectural pattern.
The critical problem in MDRE approaches, is the absenteeism of a universal and a standardized language for requirements modelling from one hand, and the lack from admittance of the supporting tools for modelling from the other hand [28]. Moreover, UML, and SysML standards are currently the most widespread used modelling languages by the Object Management Group consortium (OMG). In this work, a specific and coherent modelling language is exploited to define the requirements. This language embeds vocabulary and domain (notion) modelling into use case modelling, to guarantee better understanding and consistency of requirements specification, and it was implemented under a suitable tool support. The pet-clinic case study has been presented to test the feasibility of the model-based approach. Nonetheless, is not restricted, and can be applied for any kind of problem domain such as: physics, geography, geology, environment, e-Learning …etc.
The remaining of this paper is organized as follows: the next section gives an overview of the investigated modelling requirements language. Afterward, Section III. presents the pet-clinic example study. Finally, this paper is concluded in Section IV.

II. Methodology: from use case scenarios to CRUD implementation

A. Overview of the Requirements Specification Language

To capture the user requirements, this paper uses a consistent language called: RSL: Requirements Specific-Language. This language represents the requirements as: use case diagrams with their textual scenarios (in a constrained natural langue): i.e.: **functional requirements**, plus the domain model: **vocabulary requirements**. With the help of transformation rules, the target UML design models and code are automatically extracted. The RSL's abstract syntax is defined with the MOF standard. The requirements model is edited under a tool suite. The transformation engine includes the technological details of the envisioned platform [16].
The use case scenarios in this language are a sequence of SVO-O (Subject-Verb-Direct-Indirect Object) sentences. There are two grand categories:

1)- ***"Actor-to-" sentences***: they start with "Actor" (User) as a subject, which can perform an action such as: triggering, entering data, closing windows, …etc.
2)- ***"System-to-" sentences***: the subject is the "System". They specify reactions to the actor's actions. They serve to presenting data to the user, or performing calculations, fetching data, … and so on.
The type of the actions/reactions are distinguished according to the "*verb*". For example: Select, print, update, save, close, … etc.
In this paper, the "System-to-" sentences are used to be transformed into the Model layer code, as they contain the CRUD verbs.
The use case diagram is complemented with the vocabulary notions pertained to the problem domain.
The main domain element is the "***Concept***", that have "***Attributes***". The "***Attributes***" are linked to "***concepts***" via the "***containment***" relationship. The **"*Concepts*"** are related to each other via **"*Association*"** relationship.

B. *Transformations phases*

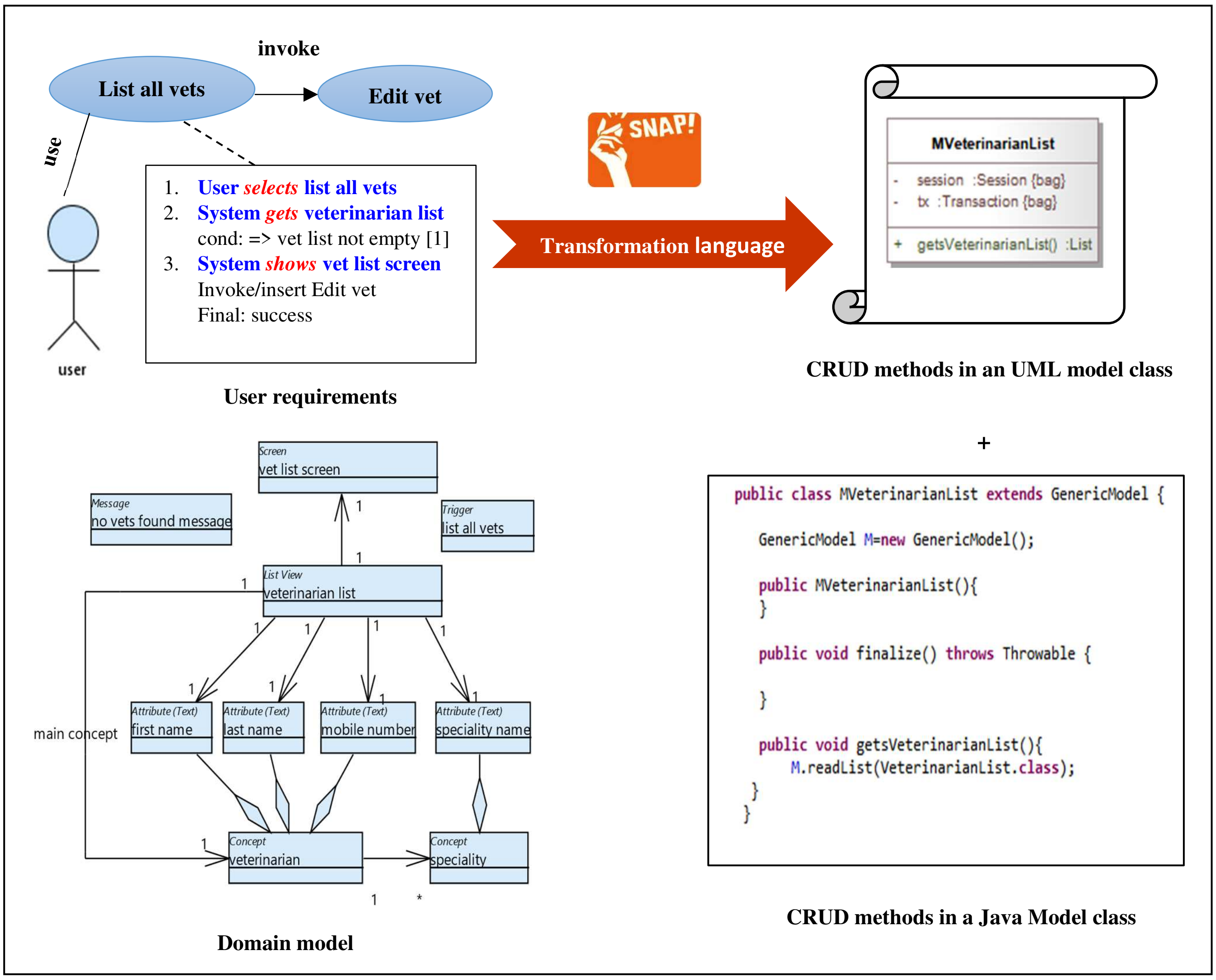
A set of transformation rules is provided to translate scenario sentences into CRUD operations. The structure of the code respects the MVC/MVP pattern. By convention, the name of the Model classes starts with “M”.
In order to derive code and UML design model, two main phases of the transformation should be pursued as follows:

- Model-to-Model transformation (M2M): to derive design models (UML class diagrams) including the CRUD operations + validate.
- Model-to-Text transformation (M2T): to extract code from the design classes for the target platform (ex. Java).

C. *Transformation rules*

- The SVO-O sentences containing the verbs pertaining to CRUD actions are translated into classes in the model layer. The name of the class is derived from the direct object (notion). For example: the “**System** ***updates*** **veterinarian data**” sentence is translated into “*MVeterinarianData*”.
- The sentences containing ***read*** verbs (read, get, fetch, retrieve … etc) are translated into operations in the appropriate model classes. The operation’s name is taken from: the verb name concatenated with the direct object. In sentence “**System** ***gets*** **pet list for owner**”, the operation’s name is “*getsPetsList*”.
- The sentences containing ***non-read*** verbs (Create, Update, Delete) are translated into operations in the appropriate model classes, with the operation’s name is taken from the verb name concatenated with the direct object. In the sentence “**System** ***saves*** **owner data**”. The name of the method is: “*savesOwnerData*” The non-crud actions can be for example: save, write, add, ... etc. (for CREATE), for UPDATE (override, edit, modify, …etc), for DELETE action (remove, erase, destroy, …etc).

Figure 1 describes the overall process of CRUD methods generation from RSL.



```
public class MVeterinarianList extends GenericModel {

   GenericModel M=new GenericModel();

   public MVeterinarianList(){
   }

   public void finalize() throws Throwable {

   }

   public void getsVeterinarianList(){
       M.readList(VeterinarianList.class);
  }
 }
```

Figure 1: From user requirements to CRUD implementation in a snap

A. *Tools used*

In this practical study, the chosen target platform was: Java. For this, the open-source Hibernate Object-Relational-Mapping (ORM) was employed, to correspond the java persistent classes (Data Transfer Objects: dtos) to the relational database code. Additional transformation rules have been developed to produce the Hibernate code (i.e.: Configuration, and mapping files). At this stage, our dedicated transformation engine that interfaces with the tool supporting RSL is utilized. Regarding, the database schema, the tables, and the views were designed under MySQL relational database management system (RDBMS) due to its popularity.
Finally, the code generator tool (here: Enterprise Architect) produces source code from the outputted UML design classes from the requirements models (also using our transformation engine). The generated code can be viewed, compiled and executed by any Java development environment (Eclipse, JBOSS, ….).

III. Pet clinic System Example study

To examine the feasibility of the given approach, we have applied it for the "***Pet Clinic***" system which is the problem domain defined using the RSL notation. This study comprehends a variety of "*user-system*" interactions, defined within use case scenarios. In the next subsections, the details for the case study are summarized.

A. *Source model*

The RSL source model is composed from: the use case diagrams with their textual scenario sentences, plus the domain models.
The use case diagram for the "***Pet clinic system***" comprises twelve use cases as indicated in Figure 2. The user is connected to the use cases via the "***use***" relationship (four use cases*: list all owners*, *add new pet*, *find pet*, and *list all vets*). The other seven use cases are interconnected through the « ***invoke*** » relationship.

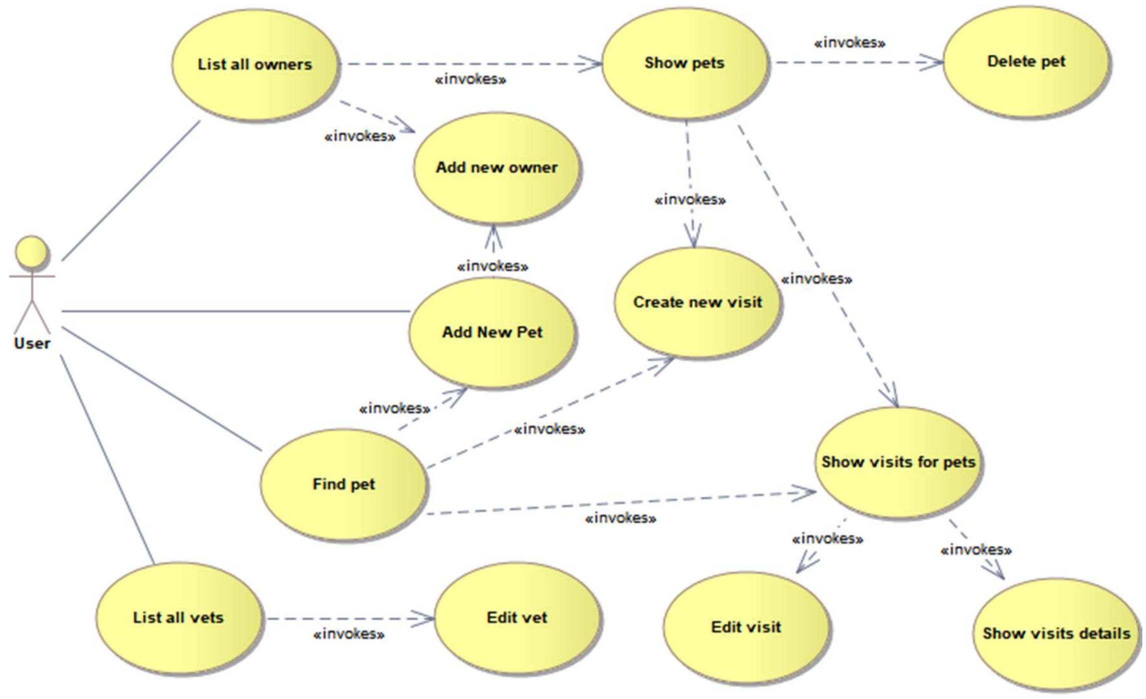
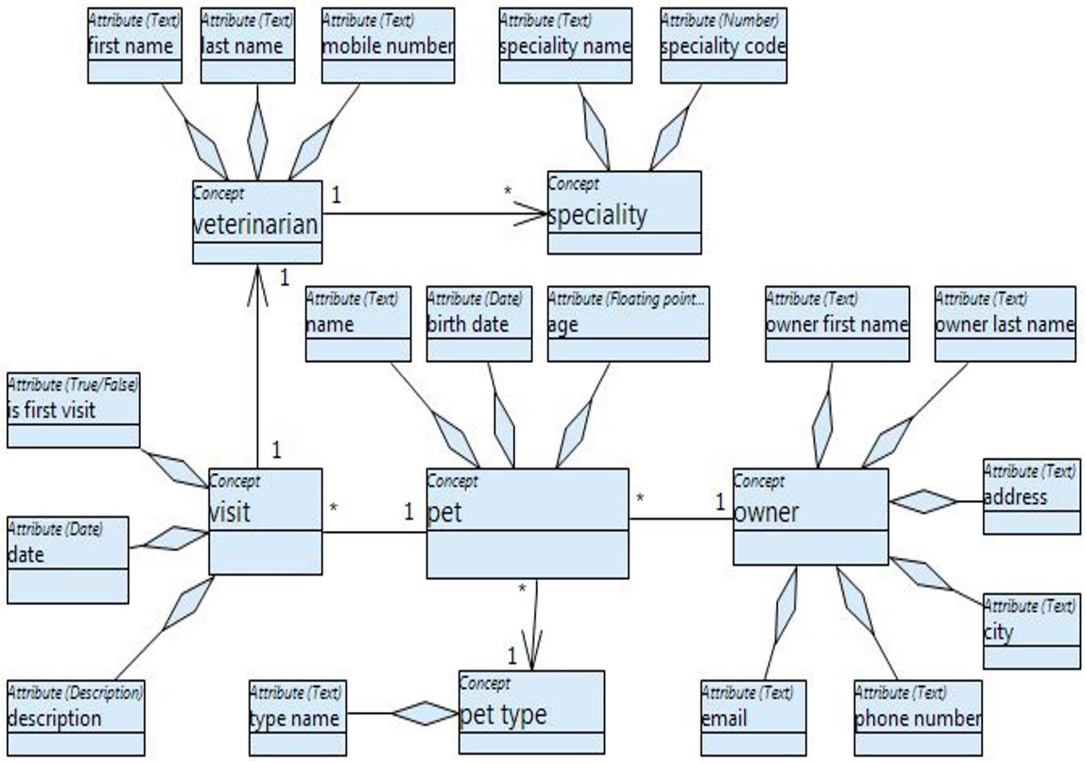


Figure 2: Use case diagram for "Pet Clinic System"

To complete the definition of the system, the "Domain Engineer" has also to develop the domain model. illustrates the main RSL domain model for the "***Pet clinic system***". It includes six concepts: "**Pet** ", "**Owner** ", "**Visit** ", "**Speciality**", "**Pet type**", and "**Veterinarian**". These concepts are interconnected trough the "***Association***" relationship, and each "concept" has "***Attributes***" linked to it with the "***Aggregation***" relationship". For example, the "Pet" concept has three attributes: "name", "age", "birthdate". The "Owner" concept has six attributes: "owner first name", "owner last name", "address", "city", "phone number", "email".
In this study, the **"*Edit visit*"** use case is described in details in
and Figure 5 respectively. The other use cases and their stories are similar to the « ***Edit visit*** » use case, and are not shown here.

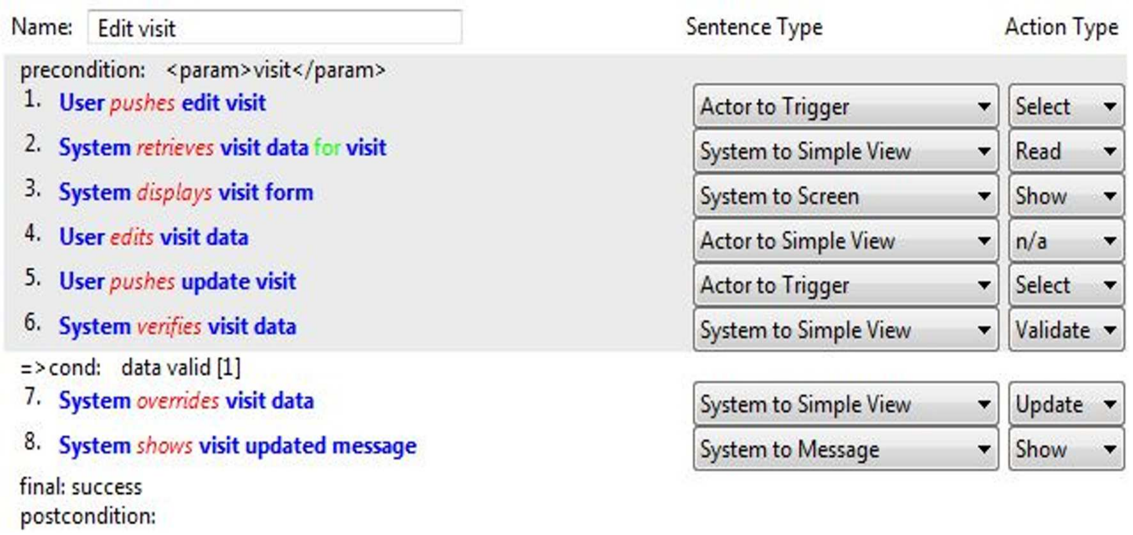
The scenario in starts with an « ***Actor-to-Trigger*** » sentence that defines the initial interaction of the user. This sentence is followed by a "***System-to-Simple View*** » sentence (**System** ***overrides*** **visit data**) to update the visit's data. Sentence no 3 is of type « ***System-to-Screen*** » to exhibit the « ***visit form*** » screen. The succeeding sentences allow to the actor to edit visit data (sentence n° 4), and to select a trigger element (here: button. Observe Sentence n°5 of Figure 5).



Attribute (Text) first name
Attribute (Text) last name
Attribute (Text) mobile number
Attribute (Text) speciality name
Attribute (Number) speciality code
Concept veterinarian
Concept speciality
1
*
1
Attribute (Text) name
Attribute (Date) birth date
Attribute (Floating point... age
Attribute (Text) owner first name
Attribute (Text) owner last name
Attribute (True/False) is first visit
1
Concept visit
*
1
Concept pet
*
1
Concept owner
Attribute (Text) address
Attribute (Date) date
*
Attribute (Text) city
1
Attribute (Description) description
Attribute (Text) type name
Concept pet type
Attribute (Text) email
Attribute (Text) phone number

Figure 3: The main RSL domain model for "Pet-clinic system"

Next, the system validates the data, if correct, the main scenario will end by two SVO sentences to update the data, and to print it in the screen for the user. Else, it will finish by also two SVO sentences to show a wrong message for the **user**, and close the visit form.



Name: Edit visit

precondition: <param>visit</param>

	Sentence Type	Action Type
1. User *pushes* edit visit	Actor to Trigger	Select
2. System *retrieves* visit data for visit	System to Simple View	Read
3. System *displays* visit form	System to Screen	Show
4. User *edits* visit data	Actor to Simple View	n/a
5. User *pushes* update visit	Actor to Trigger	Select
6. System *verifies* visit data	System to Simple View	Validate
=>cond: data valid [1]		
7. System *overrides* visit data	System to Simple View	Update
8. System *shows* visit updated message	System to Message	Show

final: success
postcondition:

Figure 4: Use case scenario for "Edit visit" use case

Figure 5 presents the domain model for "*visit data"* simple view, which is linked to a set of attributes via "1-1" relationship. Also, it is related to "*visit form*" to print the data. Other elements can be observed: "*edit visit*", and "*update visit*" triggers, and two messages: "*updated visit message*", and "*wrong visit data message*". Moreover, there is another kind of relationship between the « *visit data* » simple view and the « *visit* » concept called "**main concept**".

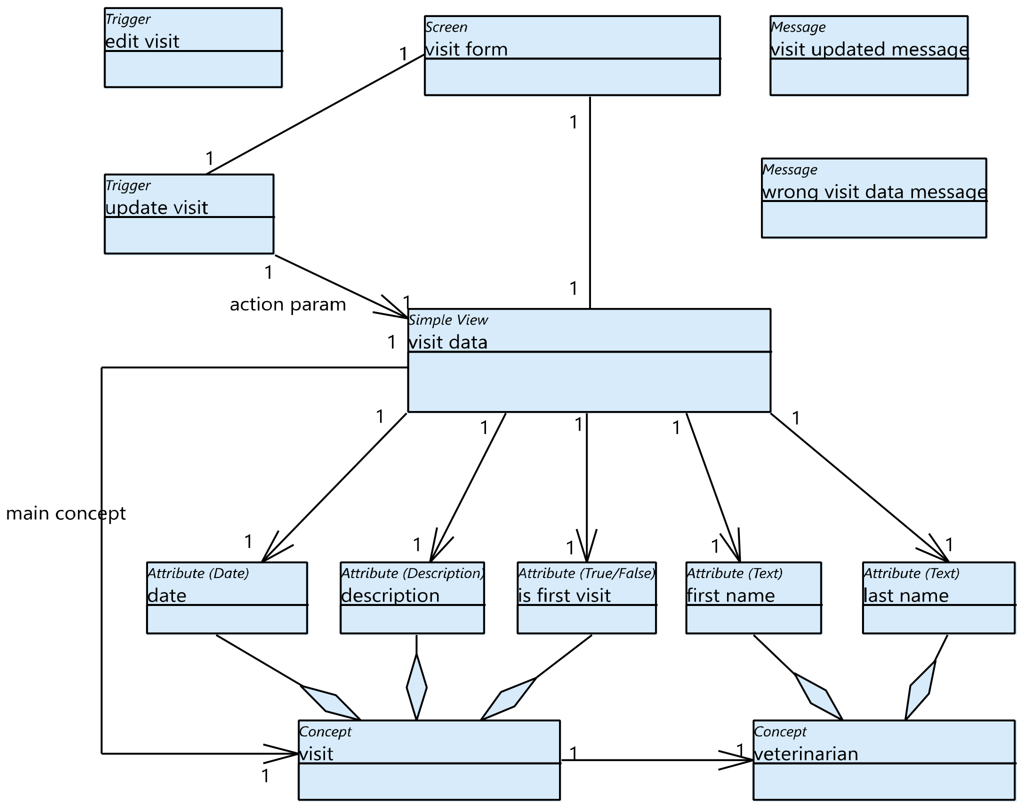


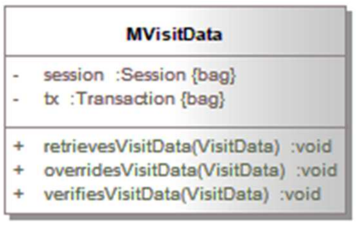
Figure 5: The source domain model for "Edit visit"

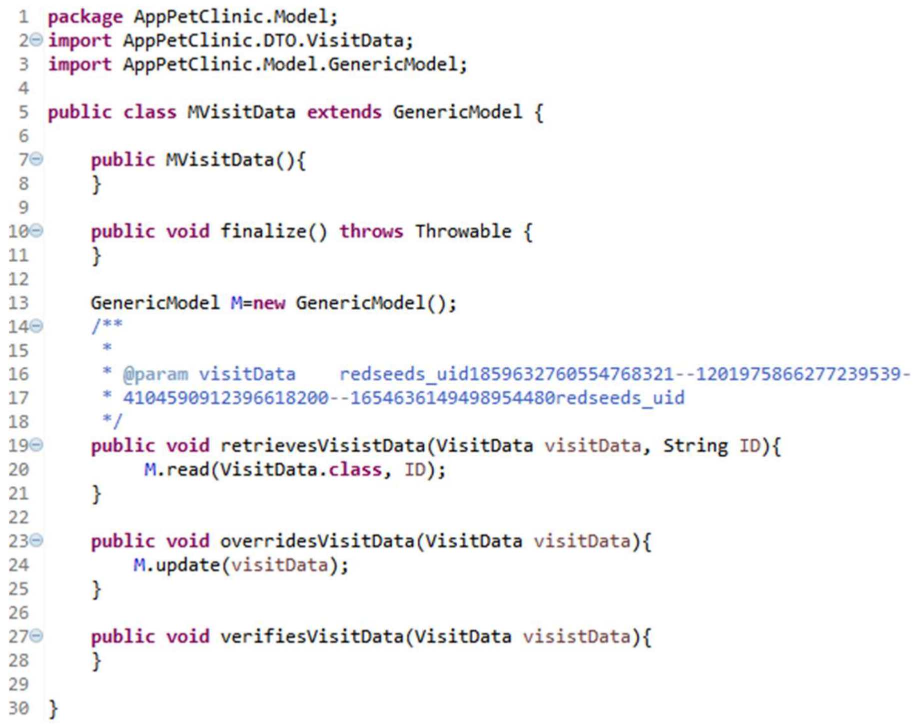
B. Target code

After finishing developing the RSL source model, the transformation program is executed, to automatically deriving the target model including UML classes, and Java code.
As aforementioned, the structure of the code adheres the MVC/MVP pattern. The code illustrated in this study reflects the Model layer for persisting and storing the data in a database.
The outputted code for the Hibernate files, and the persistent classes are not shown here.
A fragment of the resulted java code and UML class model are shown in Figure 6, for the "*Edit visit*" use case scenario.
The notion "*visit data*" (which represent the direct object in the sentences) is translated into "*MVisitData*" class: UML class (See top of Figure 6), and Java class (see bottom of Figure 6).
As can be observed in the "*Edit visit*" use case scenario in
, there exist two SVO-O sentences which comprehend two distinct CRUD verbs: sentence n° 2 contains a verb for the action "read": that is "retrieves". Sentence n°7 contains a verb for the "update" action, that is "overrides".

- The sentence "**System** ***retrieves*** **visit data**" is translated into "*retrievesVisitData*" method in "*MVisitData*" Model class. This class calls "read's" Hibernate method to fetch the data from the "*visit data*"" database view (See lines 19-21 of Figure 6).
- The sentence "**System** ***overrides*** **visit data**" is translated into "*overridesVisitData*" method in "MVisitData". This class calls the "update's" Hibernate method to update the data in the "*visit data*" database view (See lines 23-25 of Figure 6).

The code for the other scenarios, is analogous to this one, and is not given here.


MVisitData
- session :Session {bag}
- tx :Transaction {bag}
+ retrievesVisitData(VisitData) :void
+ overridesVisitData(VisitData) :void
+ verifiesVisitData(VisitData) :void



```java
package AppPetClinic.Model;
import AppPetClinic.DTO.VisitData;
import AppPetClinic.Model.GenericModel;

public class MVisitData extends GenericModel {

    public MVisitData(){
    }

    public void finalize() throws Throwable {
    }

    GenericModel M=new GenericModel();
    /**
     *
     * @param visitData    redseeds_uid1859632760554768321--1201975866277239539-
     * 4104590912396618200--1654636149498954480redseeds_uid
     */
    public void retrievesVisistData(VisitData visitData, String ID){
         M.read(VisitData.class, ID);
    }

    public void overridesVisitData(VisitData visitData){
        M.update(visitData);
    }

    public void verifiesVisitData(VisitData visistData){
    }

}
```

Figure 6: Target code: UML and Java model classes for "visit data" view

IV. Conclusion and discussion

The generation of the standard CRUD functionalities is extremely important to handle data in a database system. Modelling requirements with a consistent language allows defining correct, coherent and precise requirements. Thus, considering them as primary entities for software development, which leads to automatic extraction of code and models. In this paper, a model-driven method was applied, which focuses on modelling the initial user requirements that is Model-driven Requirements Engineering (MDRE).

A chain of translational rules has been presented in this study to automatically engender the database access implementation i.e.: CRUD operations to persist and store data in a database system. For this, we have investigated a DSL for requirements specification with use case models (functional requirements) accompanied with the domain (notion) models (vocabulary requirements). These input models are afterward transformed into a running application that complies with the popular MVC/MVP design pattern.

Furthermore, the transformation procedures should be rewritten whenever the envisioned platform changes (e.g.: web-based, mobile-based, C++, … etc.).

In order to proof the applicability of the given approach, the pet-clinic system was considered as the problem domain, and has shown the possibility of auto-generating the CRUD implementation from use case scenario sentences. Albeit, it can be applied to other problem domains such as: geography, geology, e-Learning, … etc.

From the conceptual point of view, the proposed method seems very beneficial as it makes the software development faster. Even so, it has not been analysed for industrial projects, neither for big databases. The transformation programs are capable to access and manipulate efficiency the data in the database views while applying simple requests (accessing one database view). So far, several difficulties are encountered regarding more complex "joins" queries (i.e.: manipulating multiple database views at the same time), since, these programs are very dependent to the database management system (under the theory of the DBMS must be able to…).

Acknowledgment

Many thanks to the reviewers for their attention and their important remarks for the draft of this paper. Special thanks to Prof. Michal Smilaek for his prior help regarding RSL and its supporting tool. Also, the authors would like to appreciate Prof. Kaddour Sadouni for supporting this work. The authors have no conflicts of interest to declare.

References

[1] A. Vallecillo, “On the Industrial Adoption of Model Driven Engineering. Is your company ready for MDE?,” *International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC),* vol. 1, pp. 52-68, 12 2014.

[2] S. Vaupel, G. Taentzer, R. Gerlach and M. Guckert, “Model-driven development of mobile applications for Android and iOS supporting role-based app variability,” *Software & Systems Modeling,* vol. 17, pp. 35-63, 2 2018.

[3] J. Kiswani, S. M. Dascalu, M. Muhanna and F. C. Harris, “Clowiz: A Model-driven Development Platform for Cloud-based Information Systems,” in *2018 6th International Conference on Multimedia Computing and Systems (ICMCS)*, 2018.

[4] M. Fahmideh, J. Grundy, G. Beydoun, D. Zowghi, W. Susilo and D. Mougouei, “A model-driven approach to reengineering processes in cloud computing,” *Information and Software Technology,* vol. 144, p. 106795, 2022.

[5] B. Sanden, Y. Li, J. Aker, B. Akesson, T. Bijlsma, M. Hendriks, K. Triantafyllidis, J. Verriet, J. Voeten and T. Basten, “Model-Driven System-Performance Engineering for Cyber-Physical Systems : Industry Session Paper,” in *2021 International Conference on Embedded Software (EMSOFT)*, 2021.

[6] F. Gonçalves, A. Rettberg, C. Pereira and M. Soares, “A Model-Based Engineering Methodology for Requirements and Formal Design of Embedded and Real-Time Systems,” 2017.

[7] E. Visconti, C. Tsigkanos, Z. Hu and C. Ghezzi, “Model-driven engineering city spaces via bidirectional model transformations,” *Software and Systems Modeling,* vol. 20, pp. 2003-2022, 12 2021.

[8] A. Fernández-Isabel and R. Fuentes-Fernández, “Analysis of Intelligent Transportation Systems Using Model-Driven Simulations,” *Sensors,* vol. 15, pp. 14116-14141, 2015.

[9] A. Bucchiarone and A. Cicchetti, “A Model-Driven Solution to Support Smart Mobility Planning,” in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, New York, NY, USA, 2018.

[10] M. A. Olivero, F. J. Domínguez-Mayo, C. L. Parra-Calderón, M. J. Escalona and A. Martínez-García, “Facilitating the design of HL7 domain models through a model-driven solution,” *BMC Medical Informatics and Decision Making,* vol. 20, p. 96, 5 2020.

[11] F. Ciccozzi and R. Spalazzese, MDE4IoT: Supporting the Internet of Things with Model-Driven Engineering, vol. 678, 2017, pp. 67-76.

[12] M. Mehrabi, B. Zamani and A. Hamou-Lhadj, “HealMA: a model-driven framework for automatic generation of IoT-based Android health monitoring applications,” *Automated Software Engineering,* vol. 29, p. 56, 9 2022.

[13] M. Brambilla, J. Cabot and M. Wimmer, Model-Driven Software Engineering in Practice, Second Edition, 2 ed., Springer Cham, 2017.

[14] H. Ji, O. Lenord and D. Schramm, “A Model Driven Approach for Requirements Engineering of Industrial Automation Systems,” *4th International Workshop on Equation-based Object-oriented Modeling Languages and Tools : Proceedings ; [Zurich, Switzerland, September 5, 2011 ; EOOLT 2011],* 1 2011.

[15] R. Maschotta, A. Wichmann, A. Zimmermann and K. Gruber, “Integrated Automotive Requirements Engineering with a SysML-Based Domain-Specific Language,” in *2019 IEEE International Conference on Mechatronics (ICM)*, 2019.

[16] M. Smialek and W. Nowakowski, From Requirements to Java in a Snap - Model-Driven Requirements Engineering in Practice, Springer, 2015.

[17] A. Sadovykh, D. Truscan and H. Bruneliere, “Applying Model-based Requirements Engineering in Three Large European Collaborative Projects: An Experience Report,” in *RE 2021 : 29th IEEE International Requirements Engineering Conference*, Notre Dame, South Bend, United States, 2021.

[18] R. Rodriguez-Echeverria, J. C. Preciado, Á. Rubio-Largo, J. M. Conejero, Á. E. Prieto and M. Risi, “A Pattern-Based Development Approach for Interaction Flow Modeling Language,” *Sci. Program.,* vol. 2019, 1 2019.

[19] P. Galhardo and A. R. d. Silva, “Combining Rigorous Requirements Specifications with Low-Code Platforms to Rapid Development Software Business Applications,” *Applied Sciences,* vol. 12, 2022.

[20] H. Ed-douibi, J. L. C. Izquierdo and J. Cabot, “Model-driven development of OData services: An application to relational databases,” in *2018 12th International Conference on Research Challenges in Information Science (RCIS)*, 2018.

[21] D. Sanchez, A. E. Rojas and H. Florez, “Towards a Clean Architecture for Android Apps using Model Transformations.,” *IAENG International Journal of Computer Science,* vol. 49, 2022.

[22] P. Bocciarelli, A. D’Ambrogio, T. Panetti and A. Giglio, “E-MDAV: A Framework for Developing Data-Intensive Web Applications,” *Informatics,* vol. 9, 2022.

[23] A. Deljouyi and R. Ramsin, “MDD4REST: Model-Driven Methodology for Developing RESTful Web Services,” in *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, 2022.

[24] C. Zolotas, T. Diamantopoulos, K. C. Chatzidimitriou and A. L. Symeonidis, “From requirements to source code: a Model-Driven Engineering approach for RESTful web services,” *Automated Software Engineering,* vol. 24, pp. 791-838, 12 2017.

[25] C. Zolotas, K. C. Chatzidimitriou and A. L. Symeonidis, “RESTsec: a low-code platform for generating secure by design enterprise services,” *Enterprise Information Systems,* vol. 12, pp. 1007-1033, 2018.

[26] S. M. B. A. R. K. I. M’hamed RAHMOUNI, “Model-Driven Generation of MVC2 Web Applications: From Models to Code,” *International Journal of Engineering and Applied Computer Science,* vol. 02, pp. 217-231, 7 2017.

[27] A. Moutaouakkil and S. Mbarki, “Transformation of Struts Model to Codeigniter Model,” *Procedia Computer Science,* vol. 184, pp. 767-772, 2021.

[28] E. Windisch, C. Mandel, S. Rapp, N. Bursac and A. Albers, “Approach for model-based requirements engineering for the planning of engineering generations in the agile development of mechatronic systems,” *Procedia CIRP,* vol. 109, pp. 550-555, 2022.