RM2PT: A Tool for Automated Prototype Generation from Requirements Model

Yilong Yang*, Xiaoshan Li*, Zhiming Liu†, Wei Ke‡
*Faculty of Science and Technology, University of Macau, Macau
†School of Computer and Information Science, Southwest University, China
‡Macau Polytechnic Institute, Macau
Corresponding author: Xiaoshan Li (xsl@umac.mo)

***Abstract*—Prototyping is an effective and efficient way of requirement validation to avoid introducing errors in the early stage of software development. However, manually developing a prototype of a software system requires additional efforts, which would increase the overall cost of software development. Based on our proposed approach, we develop RM2PT: a tool for generating prototypes from requirements models automatically. A requirements model consists of a use case diagram, a conceptual class diagram, system sequence diagrams for use cases, and the formal contracts of their system operations in OCL (Object Constraint Language). RM2PT can generate executable MVC (Model View Controller) prototypes from requirements models automatically. We evaluate the tool with four case studies. 93.65% of requirement specifications can be generated to the executable Java source code successfully, and only 6.35% are non-executable for our current provided generation algorithm such as sorting and event-call, which can be implemented by developers manually or invoking the APIs of advanced algorithms in Java library. The tool is efficient that the one second generated prototype of a case study requires approximate nine hours manual implementation by skilled programmers.**

The tool can be downloaded at http://rm2pt.mydreamy.net, and a demo video casting its features is at https://youtu.be/rDdpXsjSq8A

***Index Terms*—Prototype, Code Generation, Requirements Model, Requirements Validation, UML, OCL**

I. Introduction

Requirements errors are one of the causes leading failings in software projects. Careful requirements modeling along with systematic validation helps to reduce the uncertainty about target systems [1]. The goal of requirements validation is to confirm whether the requirements specification is the same as the real needs of clients. However, this process is complicated, and it is difficult to produce a consistent and complete requirements specification [2].

Rapid prototyping is an effective approach to requirements validation to demonstrate concepts, discover requirements errors and find possible fixing solutions, and discover missing requirements [3]. Besides the implementation of main system functionalities, a prototype has a User Interface (UI) [4] that allows the client to validate their requirements visually, so that faults of misunderstanding and uncertainty about the requirements can be easily found. In practice, it is very desirable to generate prototypes directly from requirements automatically with a CASE tool. However, state-of-the-art CASE tools still have long distances to reach the goal [5]. Unified Modeling Language (UML) is the de facto standard for requirement modeling and system design. Current UML modeling tools, such as Rational Rose, SmartDraw, MagicDraw, Papyrus UML, can only generate skeleton code, where classes only contain attributes and signatures of operations, not their implementations.

In this paper, we present RM2PT: a tool for automated prototype generation from a requirements model in UML diagrams complemented by OCL contracts of system operations. RM2PT has the following main features:

1) *Requirements modeling and analysis.* RM2PT contains a requirement modeler that supports bi-directionally synchronization between graphical and textual requirements models. Moreover, it provides a mechanism to identify the non-executable parts of a contract and wrap them into an interface, which can be fulfilled by developers manually or third-party APIs.
2) *Automated prototype generation from requirements model.* PM2PT contains a prototype generator that can generate prototypes from requirements models automatically. Compared with other CASE tools, RM2PT only relies on a requirements model without requiring design models. A requirements model that contains a use case diagram, a conceptual class diagram, use case definitions specified by system sequence diagrams and the OCL contracts of their system operations.
3) *Requirements validation and evolution.* Customers and developers can validate functional requirements through the generated prototypes, which provide the functions of investigating the execution processes of use cases, as well as state observation of objects, pre-condition, post-condition, and invariant checking. If some errors of requirements are found and fixed, we can regenerate a new prototype from the evolved requirements model, so that RM2PT can support the use-case driven, iterative and incremental software development process (UP) appropriately, as well as cope with requirement changes easily.

The remainder of this paper is organized as follows: Section 2 presents the RM2PT features. Section 3 presents the evaluation results on the four case studies. Section 4 and 5 discuss the related tools and conclude this paper.

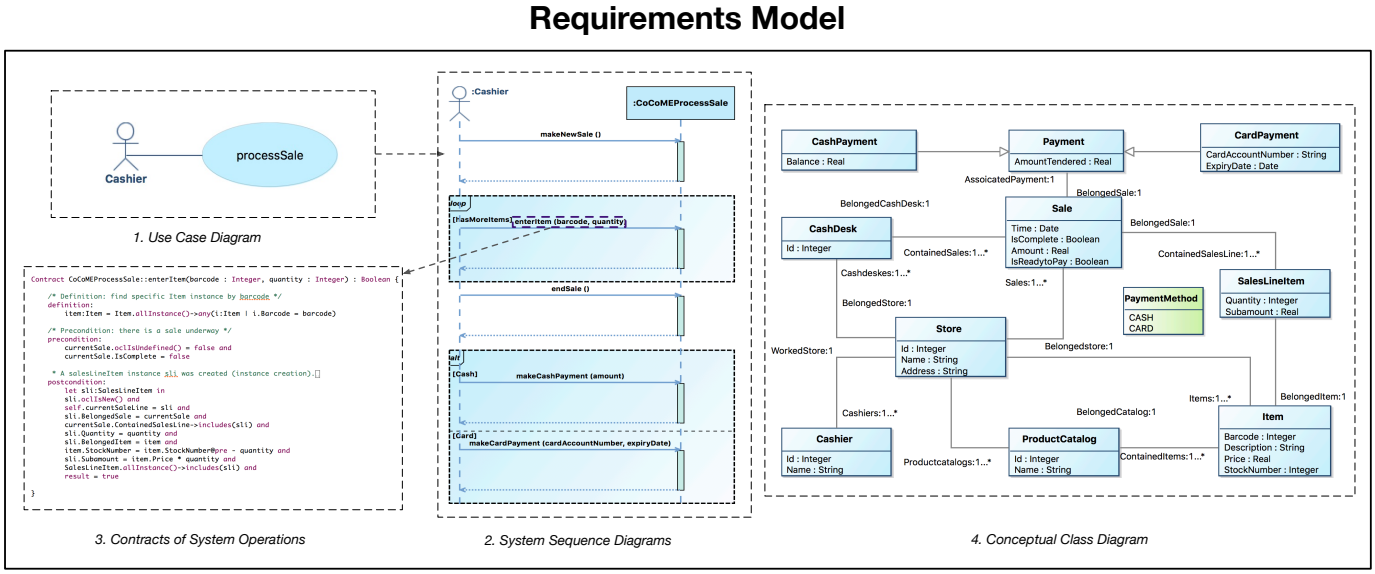

Figure 1. Requirements Model

II. RM2PT Features

A. Requirements modeling and analysis

RM2PT contains a requirement modeler and an OCL parser with bi-directionally synchronization between graphics and textual requirements models. Figure 1 is a requirements model for a supermarket system. It contains 1) a use case diagram, which captures domain processes as use cases in terms of interactions between the system and its users, 2) system sequence diagrams, which describe the interactions between actors and system of use case definitions, 3) the contracts of system operations, which specify the conditions that the state of the system is assumed to satisfy before the execution of the system operation, called the *pre-condition* and the conditions that the system state is required to satisfy after the execution, called the *post-condition* of the system operation, 4) a conceptual class diagram, which contains the conceptual entity classes and their relations of application domain. Moreover, RM2PT provides a mechanism to identify the non-executable parts of the contract, and then wrap them into an interface which can be implemented by developers manually or invoking third-party APIs. The details can be referred to our paper [6].

B. Automated prototype generation from requirements model

From a requirements model, RM2PT can automatically generate MVC prototypes, so that customers and developers can use them for requirements validation, which is shown in Figure 2. The MVC prototype contains three modules: view, controller, and model.

View: The view module contains UI widgets for customers and developers to validate use cases and observe their corresponding system state changes in Figure 3.

Controller: The controller module links the view and model modules, which makes UI events to trigger system operations.

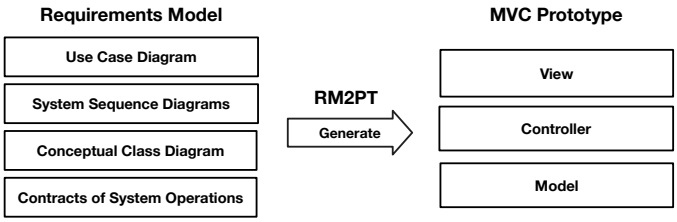

Figure 2. MVC Prototype Generation from Requirements Model

The controller listens to the events from UI widgets. When a specific event is captured, the controller retrieves the input parameters from the UI widget, and then delivers the parameters to the corresponding system operation in the model module of the prototype. Finally, the controller will update UI widgets with return results.

Model: The model module is the core of the MVC architecture pattern. It contains the classes encapsulating system operations and the classes generated from the conceptual class model. Referring to atomic actions for manipulation tables in relational databases, we introduce 13 primitive operations of object-oriented system for operation decomposition, which cover all manipulations to a) find objects, create objects, add objects, and release objects b) get and set the attributes of an object, and c) add and remove links between objects. Moreover, we introduce transformation rules and algorithms to transform the OCL contracts of system operations to primitive operations. Transformation rules are presented in this form:

$$Rule : \frac{OCL\ Expression}{Primitive\ Operation}$$

The transformation rule contains two parts: the above section is an OCL expression inside pre- or post-conditions, and the bottom part is the corresponding primitive operation. We pro-

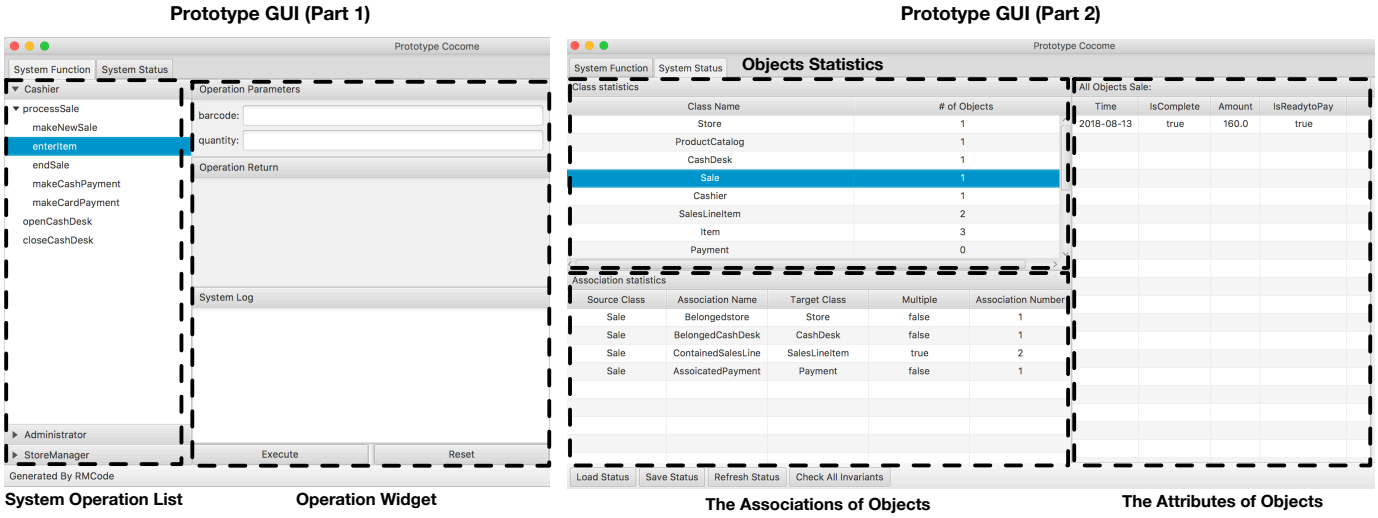

Figure 3. Prototype GUI

pose 26 transformation rules and a transformation algorithm. To implement the decomposed system operation, we first need to transform the contracts into primitive operations through the provided transformation algorithm, and then orchestrate them to be a valid system operation, finally encapsulate the system operations into classes. Based on the result, the corresponding source code of Java classes can be generated, which contains the class attributes and associations, as well as the implementation of primitive operations. The details can be referred to our paper [6].

C. Requirements Validation and Evolution

Requirements validation is to avoid introducing errors in the early stage of software development. Through the generated prototypes, customers and developers can validate functional requirements by investigating the execution processes of use cases, as well as state observation of objects, pre-condition, post-condition, and invariant checking. When executing a system operation, a warning message is prompted if a pre-condition or post-condition is not satisfied. If the execution of the system function makes system state break the related invariants, the color of the invariant bar will become red. Further inspections are required to locate the errors. The location of the errors may be in the pre-condition or post-condition of the contract. The state observation of the objects can help to locate the faults of the requirements. When clicking a class entry, the state of the corresponding attributes and associations will be displayed on the middle and left bottom side of the panel in Figure 3. Although locating and fixing the errors require more efforts, state observation and condition checking provide an intuitive way to help customers and developers to validate their requirements. More details of requirements validation and evolution are shown in our paper [6].

III. Evaluation

Four case studies are used to demonstrate the validity and capacity of RM2PT. Those case studies are widely used systems in our daily life: Supermarket System (CoCoME), Library Management System (LibMS), Automated Teller Machine (ATM), and Loan Processing System (LoanPS). The complexity of those requirements models are shown in Table I, which totally contains 17 actors, 51 use cases, 137 system operations, 980 primitive operations, 39 entity classes, 49 associations of entity classes, and 52 invariants.

Table I
The Complexity of Requirements Models

Case Study	Actor	Use Case	SO	PO	Entity Class	Association	INV
ATM	2	6	15	103	3	4	5
CoCoME	3	16	43	273	13	20	10
LibMS	7	19	45	433	11	17	25
LoanPS	5	10	34	171	12	8	12
Sum	17	51	137	980	39	49	52

* SO and PO are the abbreviations of system and primitive operations respectively. INV is the abbreviation of invariant.

The experimental settings of RM2PT are *3.5 GHz Intel Core i5, 16 GB DDR3, 500 GB Flash Storage, and JDK 8*. The results of the prototype generation are shown in Table II. On average, 93.65% system operations of those four case studies can be successfully generated without any extension. No errors are in non-executable parts, only 6.35% of them include non-executable parts in their post-conditions, which cannot be successfully generated by our currently provided generation algorithm. But they can be fixed by invoking third-party APIs or manually implementation by developers.

In practice, prototypes are manually implemented by developers according to requirements models. We compare the time cost between RM2PT and the developers for prototyping the executable parts of the requirements. Our paper [6] shows

Table II
THE GENERATION RESULT OF SYSTEM OPERATIONS

Case Study	NumSO	MSuccess	GenSuccess	SuccessRate (%)
ATM	15	15	15	100
CoCoME	43	41	40	93.02
LibMS	45	43	42	93.33
LoanPS	34	30	30	88.23
Average	34.25	32.25	31.75	93.65

* MSuccess is the number of SO which is modeled correctly without external event-call, GenSuccess is the number of SO which is successfully generated, SuccessRate = GenSuccess / NumSO.

that on average, the prototypes contain 8325 lines of code, generation spend less than 1 second, students need 17.20 hours and experienced developers require 9.14 hours.

In short, RM2PT is an efficient and effective approach to automated prototype generation than the manual prototyping. It has a) lower cost, b) no inconsistency between prototype and requirements, c) the prototypes not only contain the functions of investigating the execution processes of use cases, but also including the state observations and pre-condition, post-condition, and invariant checking. That makes customers and developers easier to find errors in their requirements. Further details are available in our paper [6].

IV. RELATED TOOLS

Most UML modeling tools support OCL-based contracts and can generate skeleton code for entity classes in the conceptual class model. The following tools are the most closely related to RM2PT. Umple [7] can generate a prototype from a class model (conceptual class diagram) and state machine models. ActionGUI [8] can generate a multi-tier application from a design model, which includes a data model (specified by ComponentUML [9]), a security model (specified by SecurityUML [9]) and a GUI model (specified by GUI Modelling Language). SCORES [10] proposed a semi-automatically approach to generating prototypes from an enhancement of the requirements specification with user interface model in FLUID [11]. MasterCraft and AndroMDA [12] can generate Java EE and .NET prototype.

Compared with RM2PT, the related works 1) require providing an explicit design model, which would increase the overall cost of software development. It contains a class diagram encapsulating system operations, the design of system operations specified in collaboration diagrams or their implementations in a programming language. In addition, SCORES and ActionGUI require a GUI design for generating the UI of prototypes. 2) Most tools start from a design model but not a requirements model, that is not the case in practical software engineering. 3) They lack the mechanism to deal with the non-executable elements in the requirements model. 4) The generated prototype from their tools does not provide the automatic mechanisms to invariants checking and object state observations in run-time for requirements validation. More details of the comparison are shown in our paper [6].

V. CONCLUSION

This paper presents the RM2PT tool based on our proposed approach to automated prototype generation from requirements models for requirements validation. The four case studies (LibMS, ATM, CoCoME and LoanPS systems) have been investigated, and the experiment result is satisfactory. The result shows that **93.65%** of use cases can be generated successfully to runnable Java source code, and **6.35%** of use cases are failure for our algorithm to generate runnable source code due to including non-executable elements in their contracts. However RM2PT can identify and then wrap them to an interface, which can be implemented by third-party APIs or developers manually.

ACKNOWLEDGMENT

This work was supported by Macau Science and Technology Development Fund (FDCT) (No. 103/2015/A3) and University of Macau (No. MYRG 2017-00141-FST), Southwest University Grant (No. SWU116007), and National Natural Science Foundation of China (NSFC) (No. 61562011, 61672435, 61732019 and 61472279)

REFERENCES

[1] A. G. Sutcliffe, A. Economou, and P. Markis, "Tracing requirements errors to problems in the requirements engineering process," *Requirements Engineering*, vol. 4, no. 3, pp. 134–151, 1999.

[2] G. Atladottir, E. T. Hvannberg, and S. Gunnarsdottir, "Comparing task practicing and prototype fidelities when applying scenario acting to elicit requirements," *Requirements Engineering*, vol. 17, no. 3, pp. 157–170, Sep. 2012.

[3] F. Kordon and Luqi, "An introduction to rapid system prototyping," *IEEE Transactions on Software Engineering*, vol. 28, no. 9, pp. 817–821, Sep. 2002.

[4] D. Baumer, W. Bischofberger, H. Lichter, and H. Zullighoven, "User interface prototyping-concepts, tools, and experience," in *Proceedings of IEEE 18th International Conference on Software Engineering (ICSE'96)*, Mar. 1996, pp. 532–541.

[5] F. Ciccozzi, I. Malavolta, and B. Selic, "Execution of UML models: a systematic review of research and practice," *Software and Systems Modeling*, Apr. 2018.

[6] Y. Yang, X. Li, Z. Liu, W. Ke, Q. Zu, and X. Chen, "Automated Prototype Generation from Formal Requirements Model," *ArXiv e-prints*, Aug. 2018.

[7] A. Forward, O. Badreddin, T. C. Lethbridge, and J. Solano, "Model-driven rapid prototyping with umple," *Software: Practice and Experience*, vol. 42, no. 7, pp. 781–797, Jul. 2012.

[8] D. Basin, M. Clavel, M. Egea, M. A. G. de Dios, and C. Dania, "A model-driven methodology for developing secure data-management applications," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 324–337, Apr. 2014.

[9] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From UML models to access control infrastructures," *ACM Transactions on Software Engineering and Methodology*, vol. 15, no. 1, pp. 39–91, Jan. 2006.

[10] A. Homrighausen, H.-W. Six, and M. Winter, "Round-trip prototyping based on integrated functional and user interface requirements specifications," *Requirements Engineering*, vol. 7, no. 1, pp. 34–45, Apr. 2002.

[11] G. Kosters, H. W. Six, and J. Voss, "Combined analysis of user interface and domain requirements," in *Proceedings of the 2th International Conference on Requirements Engineering (RE'96)*, Apr. 1996, pp. 199–207.

[12] V. Kulkarni, R. Venkatesh, and S. Reddy, "Generating enterprise applications from models," in *Proceedings of the 8th International Conference on Object-Oriented Information Systems (OOIS'02)*, Sep. 2002, pp. 270–279.