RM2Doc: A Tool for Automatic Generation of Requirements Documents from Requirements Models

Tianshu Bao
School of Computer Science and Technology
Guizhou University
Guiyang, China

Jing Yang
School of Computer Science and Technology
Guizhou University
Guiyang, China

Yilong Yang*
School of Software
Beihang University
Beijing, China

Yongfeng Yin
School of Software
Beihang University
Beijing, China

ABSTRACT

Automatic generation of requirements documents is an essential feature of the model-driven CASE tools such as UML and SysML designers. However, the quality of the generated documents from the current tools highly depends on the attached descriptions of models but not the quality of the model itself. Besides, if the stockholders ask to generate ISO/IEC/IEEE 29148-2018 conformed documents, extra templates are required. In this paper, we propose a CASE tool named RM2Doc, which can automatically generate ISO/IEC/IEEE 29148-2018 conformed requirements documents from UML models without any templates. In addition, the flow description can be generated from a use case without additional information. Moreover, it can automatically generate the semantic description of system operations only based on the formal expression of OCL. We have conducted four case studies with over 50 use cases. Overall, the result is satisfactory. The 95% requirements documents can be generated from the requirements model without any human interactions in 1 second. The proposed tools can be further developed for the industry of software engineering.

The tool can be downloaded at http://rm2pt.com/rm2doc, and a demo video casting its features is at https://youtu.be/4z0Z5mrLfBc

KEYWORDS

Automatic Documentation, Requirements, Requirements Model, Requirements Documents

ACM Reference Format:
Tianshu Bao, Jing Yang, Yilong Yang, and Yongfeng Yin. 2022. RM2Doc: A Tool for Automatic Generation of Requirements Documents from Requirements Models. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3510454.3516850

*Corresponding author: Yilong Yang (yilongyang@buaa.edu.cn)

1 INTRODUCTION

In requirements engineering, it is widely accepted that models are the preferred method of capturing and communicating requirements. Models are more formal and semantically precise than natural language, which helps developers to specify and understand requirements. However, the importance of documents should not be overlooked, as models cannot be read directly by non-specialists. In order to avoid problems in later stages, the model should be verified (to be sure developers "developed the requirements right") and validated (to be sure developers "developed the right requirements"). The verification of requirements can be achieved by automated methods[2][9]. But validation requires agreement between the developers and the stakeholders on what the system is intended to achieve[3]. A common challenge in this context is the communication gap between stakeholders and developers[7]. The communication gap between them makes stakeholders unable to participate in model validation well. Therefore, requirements documents are still required for communication and review with stakeholder.

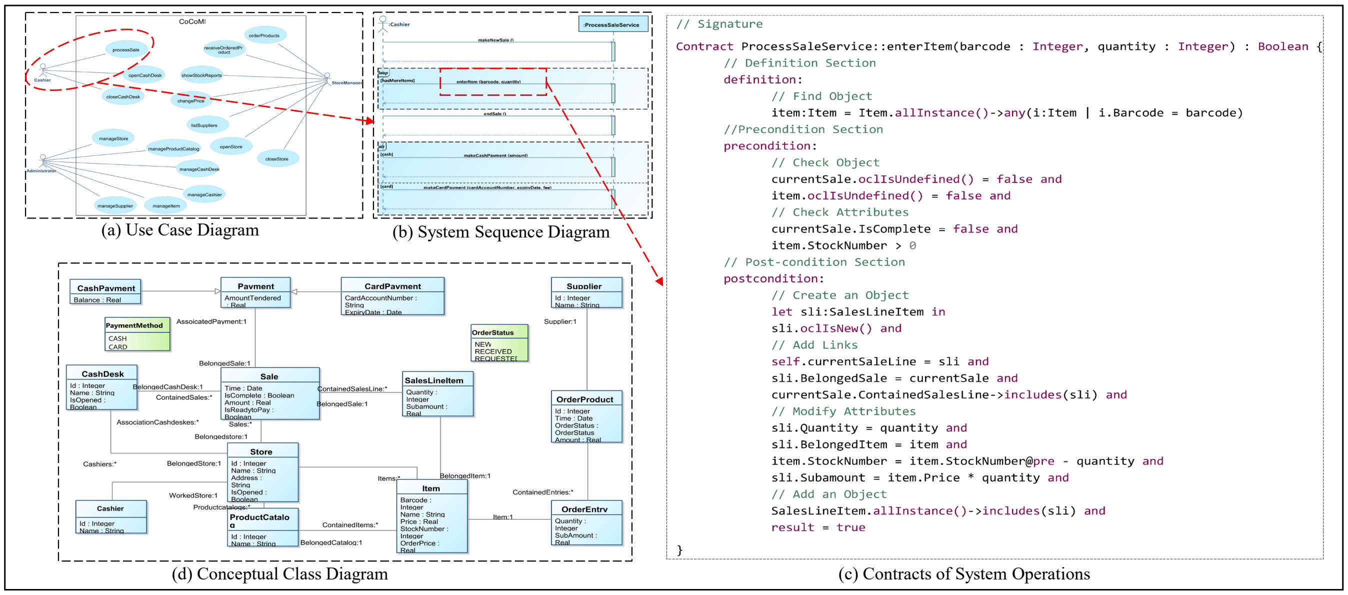
Generally, requirements documents are written manually, which is a difficult and complex task that requires a great deal of effort. This is also a factor in the lack of (good) documents for many models. Even if documents were created at an early stage, as the change of requirements and and the improvement of models, the problem of inconsistencies between the model and the document becomes more and more pronounced. To solve these problems, we present RM2Doc: a tool for automatic documents generation from a requirements model. Compared with other CASE tools, the benefits of our tool are as follows:

1) *Automatic generation of ISO/IEC/IEEE 29148-2018 conformed requirements documents from UML models without any templates.*

2) *Automatic generation of the flow description from a use case without additional information.*

3) *Automatical generation the semantic description of system operations only based on the formal expression of OCL.*

The remainder of this paper is organized as follows: Section 2 presents the overview of RM2Doc. Section 3 presents the evaluation results on the four case studies. Section 4 and 5 discuss the related tools and conclude this paper.


(a) Use Case Diagram
(b) System Sequence Diagram
(c) Contracts of System Operations
(d) Conceptual Class Diagram

```
// Signature
Contract ProcessSaleService::enterItem(barcode : Integer, quantity : Integer) : Boolean {
      // Definition Section
      definition:
            // Find Object
            item:Item = Item.allInstance()->any(i:Item | i.Barcode = barcode)
      //Precondition Section
      precondition:
            // Check Object
            currentSale.oclIsUndefined() = false and
            item.oclIsUndefined() = false and
            // Check Attributes
            currentSale.IsComplete = false and
            item.StockNumber > 0
      // Post-condition Section
      postcondition:
            // Create an Object
            let sli:SalesLineItem in
            sli.oclIsNew() and
            // Add Links
            self.currentSaleLine = sli and
            sli.BelongedSale = currentSale and
            currentSale.ContainedSalesLine->includes(sli) and
            // Modify Attributes
            sli.Quantity = quantity and
            sli.BelongedItem = item and
            item.StockNumber = item.StockNumber@pre - quantity and
            sli.Subamount = item.Price * quantity and
            // Add an Object
            SalesLineItem.allInstance()->includes(sli) and
            result = true
}
```

Figure 1: Requirements Model

2 OVERVIEW

The architecture of RM2Doc is shown in Figure 2. RM2Doc takes a requirements model as input and generates a requirements document describing that model. In this section, we first introduce our requirements modeling. Then, RM2Doc has two critical features to help stakeholders validate requirements: the first is to generate operations description from the contracts of system operations and the second is to generate the requirements document for the requirements model.

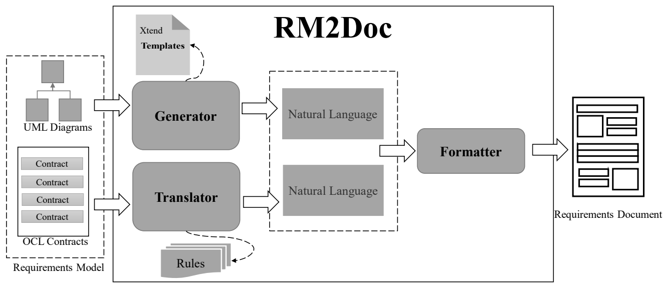


Figure 2: Overview of RM2Doc

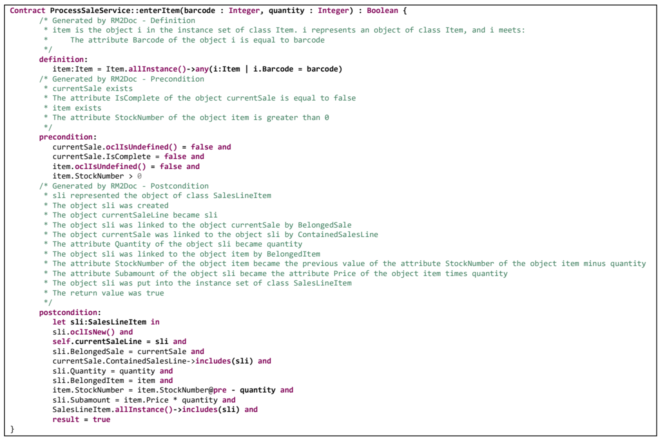
2.1 Requirements Modeling

RM2PT[11][13] is a CASE tool, which can automatically generate prototypes from formal requirements models for requirements validation. In order to provide more ways to validate requirements, in this paper we propose RM2Doc, which is an extension of the tool RM2PT. RM2Doc uses the same model as RM2PT. As shown in Figure 1, it is a lightweight formal model that contains: a use case diagram, system sequence diagrams, contracts of system operations and a conceptual class diagram.

The contract of a system operation specifies 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 (if it terminated), called the post-condition of the system operation. Figure 1 (c) shows the OCL contract of system operation *enterItem*. As shown in 1 (c), in our modeling approach, a contract consists of four parts: the signature, the definition section, the precondition section and the post-condition section:
Signature: The signature first specifies the name of the system operation and the name of the use case to which it belongs. The signature then declares the input parameters and return type of the system operation.
Definition Section: In the definition section, the objects used jointly by the precondition section and the post-condition section are defined.
Precondition Section: The precondition specifies the properties of the system state that need to be checked when system operation is to be executed. In addition to the checking of objects and attributes shown in Figure 1 (c), the precondition also includes the checking of links between objects.
Post-condition Section: The post-condition defines the possible changes that the execution of the system operation is to realize. In addition to creating and adding objects, adding links between objects, and modifying the attributes of objects as shown in Figure 2, the postconditions include the deletion of objects and the removal of links between objects.

2.2 Generation of Operations Description

A system operation is an action performed by the system in response to an external system input event. From the stakeholder's point of view, system operations define the functionality of the system. Therefore generating a description of the system operation



```
Contract ProcessSaleService::enterItem(barcode : Integer, quantity : Integer) : Boolean {
      /* Generated by RM2Doc - Definition
       * item is the object i in the instance set of class Item. i represents an object of class Item, and i meets:
       *     The attribute Barcode of the object i is equal to barcode
       */
      definition:
         item:Item = Item.allInstance()->any(i:Item | i.Barcode = barcode)
      /* Generated by RM2Doc - Precondition
       * currentSale exists
       * The attribute IsComplete of the object currentSale is equal to false
       * item exists
       * The attribute StockNumber of the object item is greater than 0
       */
      precondition:
         currentSale.oclIsUndefined() = false and
         currentSale.IsComplete = false and
         item.oclIsUndefined() = false and
         item.StockNumber > 0
      /* Generated by RM2Doc - Postcondition
       * sli represented the object of class SalesLineItem
       * The object sli was created
       * The object currentSaleLine became sli
       * The object sli was linked to the object currentSale by BelongedSale
       * The object currentSale was linked to the object sli by ContainedSalesLine
       * The attribute Quantity of the object sli became quantity
       * The object sli was linked to the object item by BelongedItem
       * The attribute StockNumber of the object item became the previous value of the attribute StockNumber of the object item minus quantity
       * The attribute Subamount of the object sli became the attribute Price of the object item times quantity
       * The object sli was put into the instance set of class SalesLineItem
       * The return value was true
       */
      postcondition:
         let sli:SalesLineItem in
         sli.oclIsNew() and
         self.currentSaleLine = sli and
         sli.BelongedSale = currentSale and
         currentSale.ContainedSalesLine->includes(sli) and
         sli.Quantity = quantity and
         sli.BelongedItem = item and
         item.StockNumber = item.StockNumber@pre - quantity and
         sli.Subamount = item.Price * quantity and
         SalesLineItem.allInstance()->includes(sli) and
         result = true
}
```

Figure 3: the Contract of System Operation *enterItem* and Semantic Description

can help the stakeholder understand the system's functionality. In this subsection, we describe how RM2Doc translates the definition section, the precondition section and the post-condition section of a contract into natural language.

As described in subsection 2.1, the definition section defines the objects. The precondition section checks the objects, their attributes and the links between objects. The post-condition section includes creation, addition, and deletion of objects, modification of attributes, and addition and deletion of links. For these different operations, we defined a total of 25 transformation rules for the three sections. Transformation rules are presented in this form:

$$Rule : \frac{OCL\ Expression}{Natural\ Language}$$

The transformation rule contains two parts: the above section is an OCL expression in the contracts, and the bottom part is the corresponding natural language.

Algorithm 1 shows how to apply the rules to translate a contract into natural language. The transformation algorithm first parses the three sections of the input *contract* into three sets of OCL sub-expressions *def*, *pre* and *post*. Then the algorithm performs rule matching on each of these three sets of expressions, translates them according to the matching result, and adds the translated result *nl* to *nls*, and finally outputs natural language *nls*. Applying the transformation rules through the transformation algorithm, the OCL contracts are translated into natural language by the translator of Figure 2.

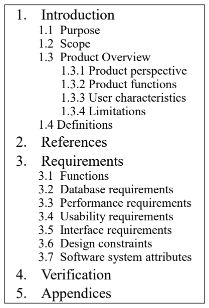
Figure 3 shows the translating result of contract of system operation *enterItem*. The natural language in the definition part and the precondition is represented in the present tense. The post-conditions represent the state of the system after execution has been completed and their natural language representation is in the past tense.

2.3 Generation of Requirements Document

Figure 5 shows the mapping from the model to the document. The product functions section and user characteristics section are generated from the use case diagram. The functional requirements

Algorithm 1: Transformation Algorithm

```
Input: Contract
Output: Natural Language
$nls \leftarrow \emptyset$;
$def \leftarrow parseDefinition(Contract)$;
$pre \leftarrow parsePrecondition(Contract)$;
$post \leftarrow parsePostcondition(Contract)$;
for $exp \in def$ do
    $num \leftarrow 0$;
    $num \leftarrow matchRule1to6(s)$;
    $nl \leftarrow translate(exp, num)$;
    $nls.append(nl)$;
end
for $exp \in pre$ do
    $num \leftarrow 0$;
    $num \leftarrow matchRule7to14(s)$;
    $nl \leftarrow translate(exp, num)$;
    $nls.append(nl)$;
end
for $exp \in post$ do
    $num \leftarrow 0$;
    $num \leftarrow matchRule15to25(s)$;
    $nl \leftarrow translate(exp, num)$;
    $nls.append(nl)$;
end
return $nls$;
```



1. Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Product Overview
      1.3.1 Product perspective
      1.3.2 Product functions
      1.3.3 User characteristics
      1.3.4 Limitations
   1.4 Definitions
2. References
3. Requirements
   3.1 Functions
   3.2 Database requirements
   3.3 Performance requirements
   3.4 Usability requirements
   3.5 Interface requirements
   3.6 Design constraints
   3.7 Software system attributes
4. Verification
5. Appendices

Figure 4: Outline

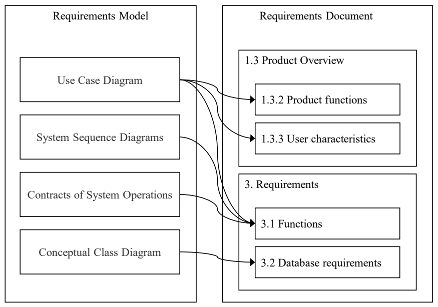
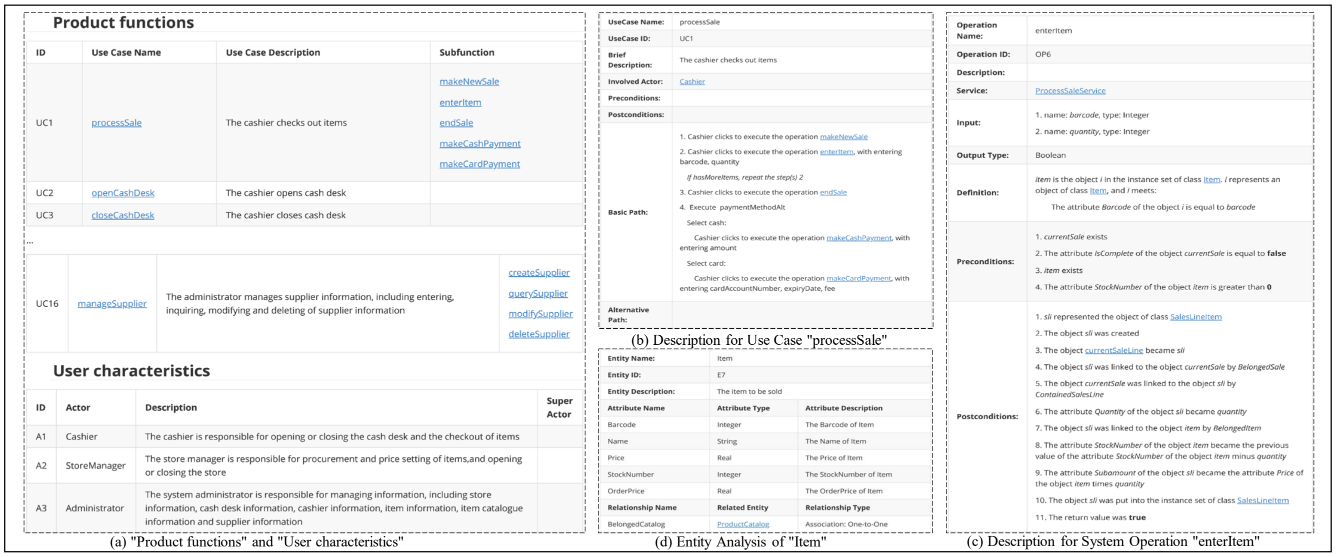


Figure 5: Mapping from Model to Document

section, which consists of a series of use case descriptions and system operation descriptions, is generated from use case diagram, system sequence diagrams and contracts of system operations. In this section, the flow description of a use case is generated by the system sequence diagram corresponding to that use case. The database requirements section is generated from the conceptual class diagram which includes a series of entity analysis.

In general, RM2Doc can generate the content of 5 parts in the requirements document: product functions, user characteristics, use case descriptions, system operation descriptions and entity analyses. For these 5 parts, we have designed five Xtend templates.



Product functions

ID	Use Case Name	Use Case Description	Subfunction
UC1	processSale	The cashier checks out items	makeNewSale enterItem endSale makeCashPayment makeCardPayment
UC2	openCashDesk	The cashier opens cash desk	
UC3	closeCashDesk	The cashier closes cash desk	

...

UC16	manageSupplier	The administrator manages supplier information, including entering, inquiring, modifying and deleting of supplier information	createSupplier querySupplier modifySupplier deleteSupplier

User characteristics

ID	Actor	Description	Super Actor
A1	Cashier	The cashier is responsible for opening or closing the cash desk and the checkout of items	
A2	StoreManager	The store manager is responsible for procurement and price setting of items,and opening or closing the store	
A3	Administrator	The system administrator is responsible for managing information, including store information, cash desk information, cashier information, item information, item catalogue information and supplier information	

(a) "Product functions" and "User characteristics"

UseCase Name:	processSale
UseCase ID:	UC1
Brief Description:	The cashier checks out items
Involved Actor:	Cashier
Preconditions:	
Postconditions:	
Basic Path:	1. Cashier clicks to execute the operation makeNewSale 2. Cashier clicks to execute the operation enterItem, with entering barcode, quantity *If hasMoreItems, repeat the step(s) 2* 3. Cashier clicks to execute the operation endSale 4. Execute paymentMethodAlt Select cash: Cashier clicks to execute the operation makeCashPayment, with entering amount Select card: Cashier clicks to execute the operation makeCardPayment, with entering cardAccountNumber, expiryDate, fee
Alternative Path:	

(b) Description for Use Case "processSale"

Entity Name:	Item	
Entity ID:	E7	
Entity Description:	The item to be sold	
Attribute Name	**Attribute Type**	**Attribute Description**
Barcode	Integer	The Barcode of Item
Name	String	The Name of Item
Price	Real	The Price of Item
StockNumber	Integer	The StockNumber of Item
OrderPrice	Real	The OrderPrice of Item
Relationship Name	**Related Entity**	**Relationship Type**
BelongedCatalog	ProductCatalog	Association: One-to-One

(d) Entity Analysis of "Item"

Operation Name:	enterItem
Operation ID:	OP6
Description:	
Service:	ProcessSaleService
Input:	1. name: *barcode*, type: Integer 2. name: *quantity*, type: Integer
Output Type:	Boolean
Definition:	*item* is the object *i* in the instance set of class Item. *i* represents an object of class Item, and *i* meets: The attribute *Barcode* of the object *i* is equal to *barcode*
Preconditions:	1. *currentSale* exists 2. The attribute *IsComplete* of the object *currentSale* is equal to **false** 3. *item* exists 4. The attribute *StockNumber* of the object *item* is greater than **0**
Postconditions:	1. *sli* represented the object of class SalesLineItem 2. The object *sli* was created 3. The object currentSaleLine became *sli* 4. The object *sli* was linked to the object *currentSale* by *BelongedSale* 5. The object *currentSale* was linked to the object *sli* by *ContainedSalesLine* 6. The attribute *Quantity* of the object *sli* became *quantity* 7. The object *sli* was linked to the object *item* by *BelongedItem* 8. The attribute *StockNumber* of the object *item* became the previous value of the attribute *StockNumber* of the object *item* minus *quantity* 9. The attribute *Subamount* of the object *sli* became the attribute *Price* of the object *item* times *quantity* 10. The object *sli* was put into the instance set of class SalesLineItem 11. The return value was **true**

(c) Description for System Operation "enterItem"

Figure 6: Requirements Document

By using these 5 Xtend templates, RM2Doc can generate a requirements document from a requirements model. In the rest of the requirements document, we have predefined guidelines for writing in accordance with the ISO/IEC/IEEE 29148-2018.

3 EVALUATION

We will introduction case studies first, then propose the evaluation results based on the case studies.

3.1 Case Studies

In order to evaluate the RM2Doc, we reuse four case studies from the paper [12]. 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). More details of the requirements models can be found at GitHub[1]. The complexity of those requirements models are shown in Table 1, which totally contains 17 actors, 51 use cases, 6 system sequence diagrams, 138 system operations, 38 entity classes, and 49 associations of entity classes.

Table 1: The Complexity of Requirements Models

Case Study	Actor	Use Case	SSD	SO	Entity Class	Association
ATM	2	6	3	15	2	4
CoCoME	3	16	1	43	13	20
LibMS	7	19	0	45	11	17
LoanPS	5	10	2	35	12	8
Sum	17	51	6	138	38	49

* SSD is the abbreviation of system sequence diagram. SO is the abbreviations of system operations.

[1]https://github.com/RM2PT/CaseStudies

To illustrate the capabilities of RM2Doc, this section demonstrates the generation of a requirements document from a requirements model. We use CoCoME as an example. The use case diagram in Figure 1 (a), contains 3 participants and 16 use cases. RM2Doc generates the "Product functions" and the "User characteristics" of the document, as shown in Figure 6 (a).

Considering the system sequence diagram and the use case diagram in Figure 1, RM2Doc can generate a use case description for the use case *processSale*, as shown in Figure 6 (b). Figure 1 (c) presents the contract of *enterItem*, which is a system operation in Figure 1 (b). According to the contract, RM2Doc generates a description of the system operation *enterItem*, as shown in Figure 6 (c). In total, 16 use case descriptions and 43 system operation descriptions are generated, and they form the "Functional requirements" of the document. Based on CoCoME's conceptual class diagram, as shown in Figure 1 (d), RM2Doc generates the analysis of 13 entities, which are then organized into the "Database requirements" of the document. The Figure 6 (d) shows the generation result of the entity *Item*.

3.2 Evaluation Results

Table 2: The Generation Result of Operations Description

Case Study	NumSO	MSuccess	GenSuccess	SuccessRate (%)
ATM	15	15	15	100
CoCoME	43	42	41	95.34
LibMS	45	44	42	93.33
LoanPS	35	32	32	91.42
Total	138	133	130	95.02

* 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.

Correctness of generation. For the correctness of the generation, we consider two aspects. 1) For the generation of UML diagram results, 17 actors, 51 use cases, 6 system sequence diagrams, 38 entity classes, and 49 associations of entity classes were all generated correctly. 2) As for the OCL contracts, the results of 138 system operations are shown in Table 2, of which, 133 system operations can be modeled correctly. And 130 of the 133 can generate semantic descriptions correctly. The average success rate of the four cases was 95.02%.
Time performance. The main application scenario for RM2Doc's documents generation is to provide natural language reference for stakeholders. In addition, our generation can also provide developers with alternative views of the model. In the latter case, the generation must be fast enough[7]. The experimental settings of RM2Doc are 2.8 GHz Intel Core i5, 8 GB DDR3, and JDK 11. Generation for all 4 case studies spend less than 1 second, we consider that this speed meets our expectations.

4 RELATED TOOLS

The tools closely related to RM2Doc can be divided into two categories according to the results generated.

The tools in the first category, which generate natural language requirements from UML models, are as follows. TESSI[6] generates natural language text for describing the model based on the UML model. GeNLangUML[8] and KeY[1] use UML class diagrams as input to generate natural language descriptions of class diagrams. In addition, there has been some work[3][4] dedicated to extracting well-structured business vocabulary and business rules from use case diagrams and class diagrams. These tools use a single type of model, and the results are mostly a set of disordered sentences or not well organised. It is very difficult to find valuable content from such results, and the application value is low.

The second type of tool is focused on generating the document from software models. Wang et al.[10] propose a tool that generates software requirement specifications, preliminary design specifications and detailed design specifications from UML models and SmartC models. DocGen[5] generates a report from SysML models. The modelling tools MagicDraw[2] and Modelio[3] have an integrated documents generation function. In addition, some commercial tools such as M2Doc[4], GenDoc[5] and pxDoc[6] also support the generation of structural documents from models. Such tools can generate the organisation of the document, but 1) the content in the document relies too much on the natural language descriptions that come with the model, rather than on natural language generation techniques. 2) Most tools require a template for the document. Some templates are quite complex to produce and for the novice this can be costly. 3) All these tools do not include the handling of OCL, which is a complement to UML.

Compared to the above tools, RM2Doc 1) can generate well-structured documents without additional templates from the user. 2) The natural language in the document does not only rely on the natural language descriptions that come with the model, but can also be generated by the model, e.g., a flow of a use case can be generated from the system sequence diagram of the use case. 3) Our tool can also translate the preconditions and post-conditions of system operations into natural language.

[2]https://www.magicdraw.com/main.php
[3]https://www.modeliosoft.com/en
[4]https://www.m2doc.org
[5]https://www.eclipse.org/gendoc
[6]https://www.pxdoc.fr

5 CONCLUSION

This paper presents the RM2Doc tool. The tool generates requirements documents from the requirements model and improves the productivity of requirements documents. The generated requirements document helps the stakeholders to perform requirements validation. Four case studies evaluated the capability of RM2Doc.

In the future, we will make some improvements to make our tools more useful. For example, we plan use deep learning to generate more flexible and more readable nature language from requirements model.

ACKNOWLEDGMENTS

This work was supported by National Science Foundation of China Grant NO. 61732019, and the Opening Project of Shanghai Trusted Industrail Control Platform No. KH54327801

REFERENCES

[1] Håkan Burden and Rogardt Heldal. 2011. Natural Language Generation from Class Diagrams. In *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa)*. Association for Computing Machinery, 1–8.

[2] Jordi Cabot, Robert Clarisó, and Daniel Riera. 2007. UMLtoCSP: A Tool for the Formal Verification of UML/OCL Models Using Constraint Programming. In *Proceedings of the 22th IEEE/ACM International Conference on Automated Software Engineering (ASE '07)*. Association for Computing Machinery, 547–548.

[3] Jordi Cabot, Raquel Pau, and Ruth Raventós. 2010. From UML/OCL to SBVR specifications: A challenging transformation. *Information systems* 35, 4 (2010), 417–440.

[4] Paulius Danenas, Tomas Skersys, and Rimantas Butleris. 2020. Natural language processing-enhanced extraction of SBVR business vocabularies and business rules from UML use case diagrams. *Data Knowl. Eng.* 128 (2020), 101822.

[5] Christopher Delp, Doris Lam, Elyse Fosse, and Cin-Young Lee. 2013. Model based document and report generation for systems engineering. In *2013 IEEE Aerospace Conference*. IEEE, 1–11.

[6] Petr Kroha, Philipp Gerber, and Lars Rosenhainer. 2006. Towards generation of textual requirements descriptions from UML models. In *Proceedings of the 9th International Conference Information Systems Implementation and Modelling (ISIM '2006)*. 31–38.

[7] Henrik Leopold, Jan Mendling, and Artem Polyvyanyy. 2014. Supporting Process Model Validation through Natural Language Generation. *IEEE Transactions on Software Engineering* 40, 8 (Aug 2014), 818–840.

[8] Farid Meziane, Nikos Athanasakis, and Sophia Ananiadou. 2008. Generating natural language specifications from UML class diagrams. *Requirements Engineering* 13, 1 (2008), 1–18.

[9] Aurelijus Morkevicius and Nerijus Jankevicius. 2015. An approach: SysML-based automated requirements verification. In *2015 IEEE International Symposium on Systems Engineering (ISSE '15)*. IEEE, 92–97.

[10] Chao Wang, Hong Li, Zhigang Gao, Min Yao, and Yuhao Yang. 2010. An automatic documentation generator based on model-driven techniques. In *2010 2nd International Conference on Computer Engineering and Technology*, Vol. 4. IEEE, V4–175–V4–179.

[11] Yilong Yang, Wei Ke, and Xiaoshan Li. 2019. RM2PT: Requirements Validation through Automatic Prototyping. In *27th IEEE International Requirements Engineering Conference, RE 2019, Jeju Island, Korea (South), September 23-27, 2019*. IEEE, 484–485.

[12] Yilong Yang, Xiaoshan Li, Wei Ke, and Zhiming Liu. 2020. Automated Prototype Generation From Formal Requirements Model. *IEEE Transactions on Reliability* 69, 2 (2020), 632–656.

[13] Yilong Yang, Xiaoshan Li, Zhiming Liu, and Wei Ke. 2019. RM2PT: A Tool for Automated Prototype Generation from Requirements Model. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE '19)*. IEEE, 59–62.