RM2PT: Automated Prototype Generation from Requirements Model

Yilong Yang

Department of Computer Science and Information
Faculty of Science and Technology
University of Macau
Macau. China
yylonly@gmail.com

June 18, 2019

Contents

Contents

Motivation

- Rapid prototyping is an effective and efficient way for requirements validation.
- However, manually developing a prototype would increase the overall cost of software development.
- It is very desirable to have an approach and a CASE tool that can automatically generate prototypes directly from requirements.

Related Work

- Current UML modeling tools can only generate skeleton code, where classes only contain attributes and operation signatures, not their implementations.
- To generate prototypes, a design model is required, which contains how to encapsulate system operations into classes and how to collaborate objects to fulfill system operations.
- They lack the mechanism to deal with the non-executable elements in the requirements model.
- The generated prototype does not provide the automatic mechanisms in run-time to consistency checking and state observations for requirements validation.

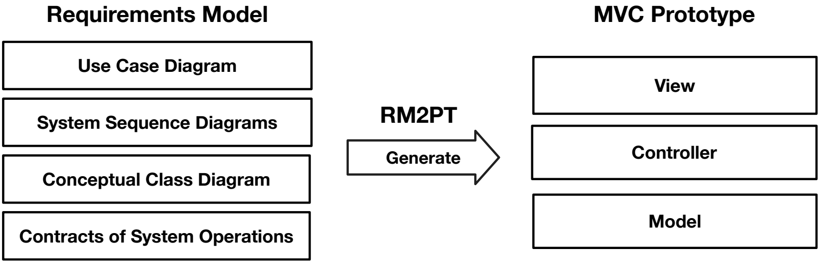
Contribution

We introduce a CASE tool for generating prototypes automatically, which

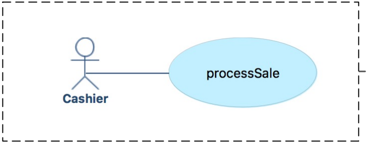
- do not require design models but only rely on a requirements model
- provide a mechanism to identify the non-executable parts of a contract and wrap them into an interface, which can be fulfilled by developers manually or third-party APIs
- contain validity and consistency checking as well as state observation in the generated prototypes

Contents

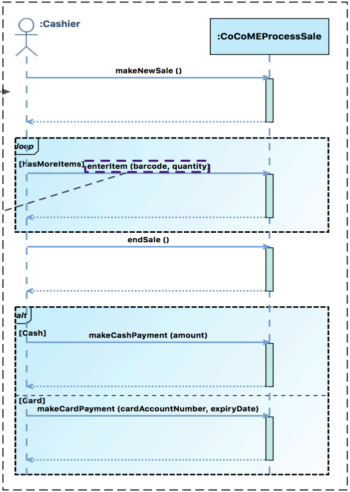
Overview


Requirements Model
Use Case Diagram
System Sequence Diagrams
Conceptual Class Diagram
Contracts of System Operations
RM2PT
Generate
MVC Prototype
View
Controller
Model

Requirements Model



1. Use Case Diagram

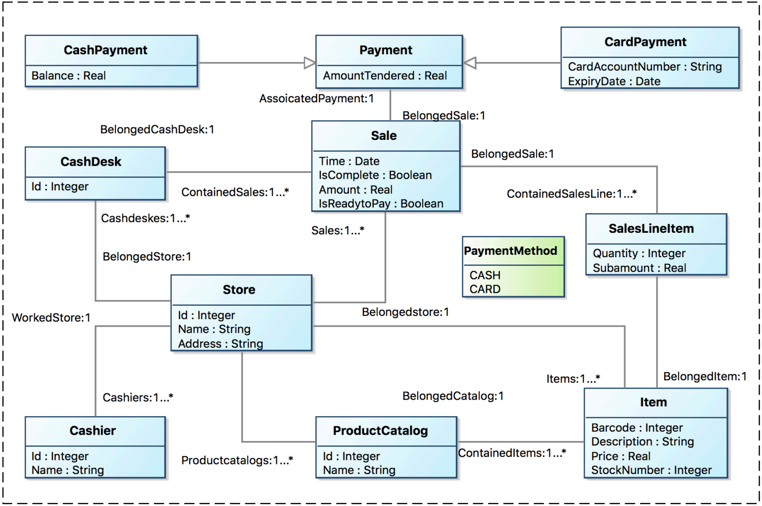


2. System Sequence Diagrams

```
Contract CoCoMEProcessSale::enterItem(barcode : Integer, quantity : Integer) : Boolean {

    /* Definition: find specific Item instance by barcode */
    definition:
        item:Item = Item.allInstance()->any(i:Item | i.Barcode = barcode)

    /* Precondition: there is a sale underway */
    precondition:
        currentSale.oclIsUndefined() = false and
        currentSale.IsComplete = false

     * A salesLineItem instance sli was created (instance creation).
    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

}
```

3. Contracts of System Operations

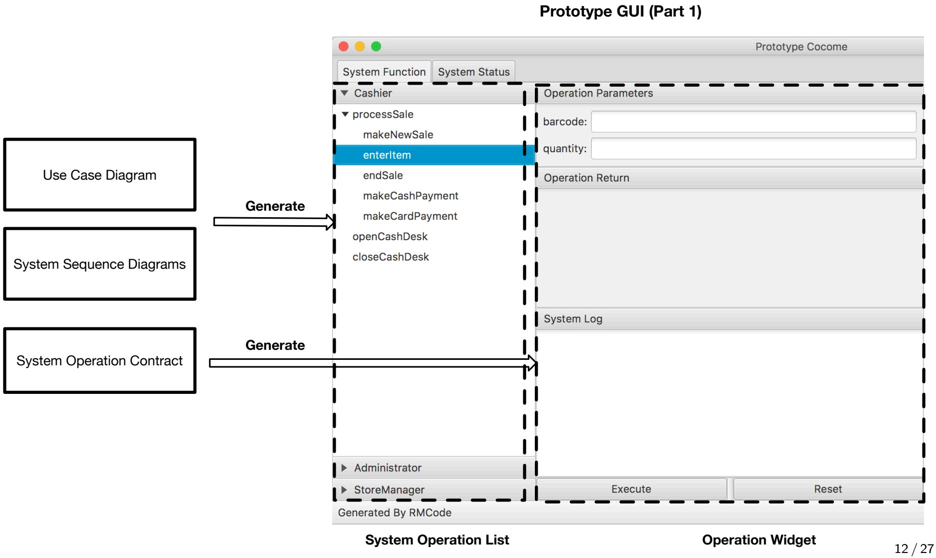
Requirements Model



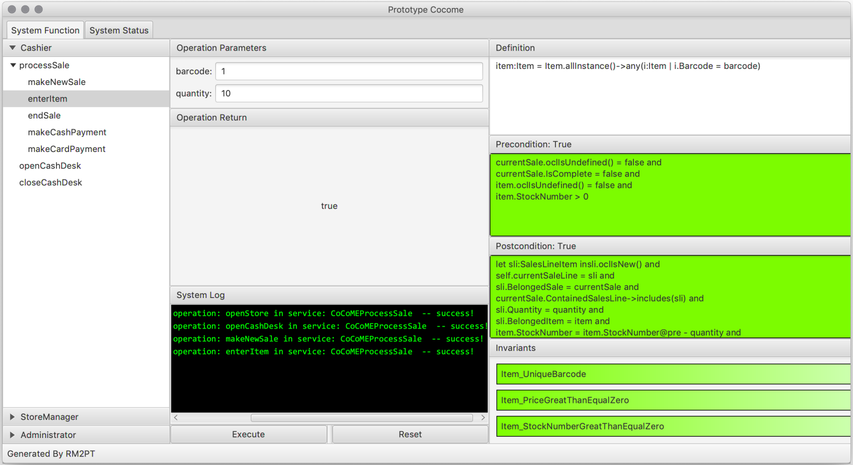
4. Conceptual Class Diagram

Contents

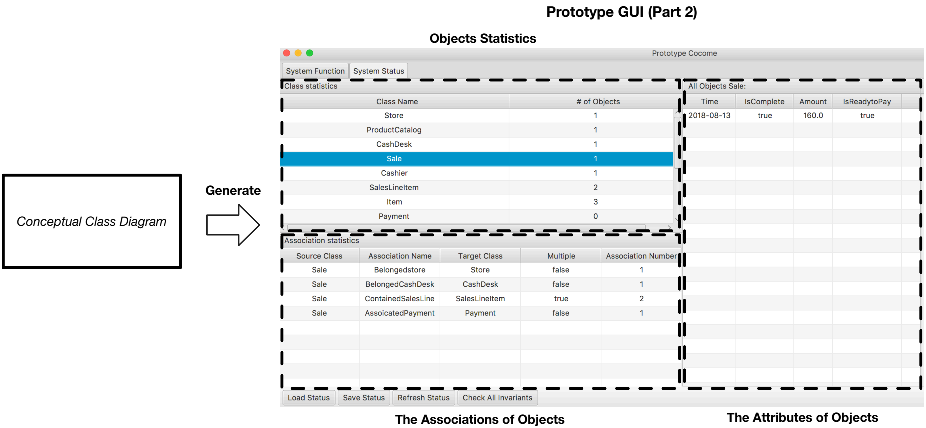
Prototype GUI (Execution)



Prototype GUI (Execution)



Prototype GUI (Observation)



Contents

Case Studies

- ATM - Automated Teller Machine
- CoCoME - Supermarket System
- LibMS - Library Management System
- LoanPS - Loan Processing System

Complexity of Requirements Models

Table 1: The Complexity of Requirements Models

Case Study	Actor	Use Case	SO	AO	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

* Above table shows the number of elements in the requirements model. SO and AO are the abbreviations of system and primitive operations respectively. INV is the abbreviation of invariant.

Cost of Requirements Modeling

Table 2: Cost of Requirements Modeling

Case Study	UML Diagram	OCL Contracts	Total (hours)
ATM	1.01	1.32	2.33
CoCoME	4.55	4.91	9.46
LibMS	4.64	6.37	11.01
LoanPS	5.51	6.94	12.45
Average	3.92	4.88	8.81

* UML diagram contains a use case diagram, system sequence diagrams, and a conceptual class diagram.

Generation Result of System Operations

Table 3: 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.

Automated Prototyping vs Manual Prototyping

Table 4: Manual Prototyping

Case Study	Implementation	Testing	Debugging	Total (hr)
ATM	6.09	4.63	3.90	14.62
CoCoME	15.08	8.80	8.31	32.19
LibMS	18.28	9.18	7.29	34.74
LoanPS	13.23	8.96	8.79	30.98
Average	13.17	7.89	7.07	28.13

Automated Prototyping vs Manual Prototyping

Table 5: Automated Prototyping

Name	Line of Code	Automated Prototype (ms)	System Operation (ms)
ATM	3897	309.74	2.26
CoCoME	9572	788.99	9.78
LibMS	12017	1443.39	18.22
LoanPS	7814	832.78	5.52
Average	8325	843.73	8.95

Scope and Limitation

Our approach has the scopes of application for practical problems.

- The requirements model and the generated prototypes of our approach are object-oriented.
- Our approach suitable for modeling and validating object-oriented information systems, enterprise systems, and interactive systems. The batching systems have heavy internal workloads are not suited for.
- Moreover, our approach focuses on functional requirements but not non-functional requirements such as time, dependability, security, and space. That means the real-time systems, embedding systems, and cyber-physical systems are not suitable for our approach.

Contents

Conclusion

We presents a CASE tool to automated prototype generation from a requirements model.

- The executable parts of the contract are translated into Java source code. The non-executable parts of a contract can be identified and wrapped by an interface, which can be fulfilled by third-party APIs.
- Four cases studies have been investigated, and the experiment result is satisfactory that the **93%** of system operations of use cases can be generated successfully in 1 second.

Future Work

- Improve the current transformation algorithm to cover the more substantial subset of the executable specification.
- Integrate current prototyping tool with our another work on automated translating use case definitions in natural language into their corresponding formal contract in OCL.
- Furthermore, after a system requirements model is validated by prototyping, we plan to generate the prototype into its corresponding real system.

RM2PT

THANK YOU