1 Introduction

1.1 Purpose

This subsection should

- a) Delineate the purpose of the SRS;
- b) Specify the intended audience for the SRS.

1.2 Scope

Name of software to be developed: Takeout System

This subsection should

- b) Explain what the software product(s) will, and, if necessary, will not do;
- c) Describe the application of the software being specifified, including relevant benefifits, objectives, and goals;
- d) Be consistent with similar statements in higher-level specififications (e.g., the system requirements specifification), if they exist.

1.3 Product Overview

1.3.1 Product perspective

This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.

This subsection should also describe how the software operates inside various constraints. For example,
these constraints could include

- a) System interfaces;
- b) User interfaces;
- c) Hardware interfaces;
- d) Software interfaces;
- e) Communications interfaces;
- f) Memory;
- j) Operations;
- k) Site adaptation requirements.

1.3.1.1 System interfaces

SI1 - TakeoutSystem

Service Name:	TakeoutSystem
Service ID:	SI1
Description:	
Operation:	• search • enterStore • excursionPublicOrder • acceptOrder • terminateOrder
Temporary Variable	**Variable Description**
CurrentStore	CurrentStore is a object of Store
CurrentDilivery	CurrentDilivery is a object of Dilivery

SI2 - ThirdPartyServices

Service Name:	ThirdPartyServices
Service ID:	SI2
Description:	
Operation:	

SI3 - ProcessOrderService

Service Name:	ProcessOrderService
Service ID:	SI3
Description:	
Operation:	• makeNewOrder • enterItem • endOrder • makeCashPayment • makeCardPayment
Temporary Variable	**Variable Description**
CurrentOrderLine	CurrentOrderLine is a object of OrderLineItem
CurrentSale	CurrentSale is a object of Sale
CurrentPaymentMethod	CurrentPaymentMethod has several options: [CASH|CARD]

SI4 - ManageItemCRUDService

Service Name:	ManageItemCRUDService
Service ID:	SI4
Description:	
Operation:	• createItem • queryItem • modifyItem • deleteItem

SI5 - ManageStoreCRUDService

Service Name:	ManageStoreCRUDService
Service ID:	SI5
Description:	
Operation:	• createStore • queryStore • modifyStore • deleteStore

SI6 - ManageDiliveryCRUDService

Service Name:	ManageDiliveryCRUDService
Service ID:	SI6
Description:	
Operation:	• createDilivery

1.3.2 Product functions

Use Case Diagram


Use Case Diagram

ID	Use Case Name	Use Case Description	Subfunction
UC1	search		
UC2	checkOrder		
UC3	publicOrderInfo		
UC4	acceptOrder		
UC5	ercommendOrder		
UC6	excursionPublicOrder		
UC7	manageStore		createStore queryStore modifyStore deleteStore
UC8	manageDilivery		createDilivery
UC9	manageCustomer		
UC10	processOrder		makeNewOrder enterItem endOrder makeCashPayment makeCardPayment
UC11	manageItem		createItem queryItem modifyItem deleteItem
UC12	enterStore		
UC13	terminateOrder		

1.3.3 User characteristics

ID	Actor	Description	Super Actor
A1	Customer		
A2	Business		
A3	Administer		
A4	Dilivery		

1.3.4 Limitations

This subsection of the SRS should provide a general description of any other items that will limit the developer's options. These include

- a) Regulatory policies;
- b) Hardware limitations (e.g., signal timing requirements);
- c) Interfaces to other applications;
- d) Parallel operation;
- e) Audit functions;
- f) Control functions;
- g) Higher-order language requirements;
- h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);
- i) Reliability requirements;
- j) Criticality of the application;
- k) Safety and security considerations.
- l) physical/mental considerations; and
- m) limitations that are sourced from other systems, including real-time requirements from the controlled system through interfaces.

1.4 Definitions

This subsection should provide the defifinitions of all terms required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

2 References

This subsection should

- a) Provide a complete list of all documents referenced elsewhere in the SRS;
- b) Identify each document by title, report number (if applicable), date, and publishing organization;
- c) Specify the sources from which the references can be obtained.

This information may be provided by reference to an appendix or to another document.

3 Requirements

3.1 Functions

3.1.1 Use Case

UC1 - search

Use Case Description:

UseCase Name:	search
UseCase ID:	UC1
Brief Description:	
Involved Actor:	Customer
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC2 - checkOrder

Use Case Description:

UseCase Name:	checkOrder
UseCase ID:	UC2
Brief Description:	
Involved Actor:	CustomerBusiness
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC3 - publicOrderInfo

Use Case Description:

UseCase Name:	publicOrderInfo
UseCase ID:	UC3
Brief Description:	
Involved Actor:	Business
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC4 - acceptOrder

Use Case Description:

UseCase Name:	acceptOrder
UseCase ID:	UC4
Brief Description:	
Involved Actor:	Dilivery
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC5 - ercommendOrder

Use Case Description:

UseCase Name:	ercommendOrder
UseCase ID:	UC5
Brief Description:	
Involved Actor:	Dilivery
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC6 - excursionPublicOrder

Use Case Description:

UseCase Name:	excursionPublicOrder
UseCase ID:	UC6
Brief Description:	
Involved Actor:	Dilivery
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC7 - manageStore

Use Case Description:

UseCase Name:	manageStore
UseCase ID:	UC7
Brief Description:	
Involved Actor:	Administer
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC8 - manageDilivery

Use Case Description:

UseCase Name:	manageDilivery
UseCase ID:	UC8
Brief Description:	
Involved Actor:	Administer
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

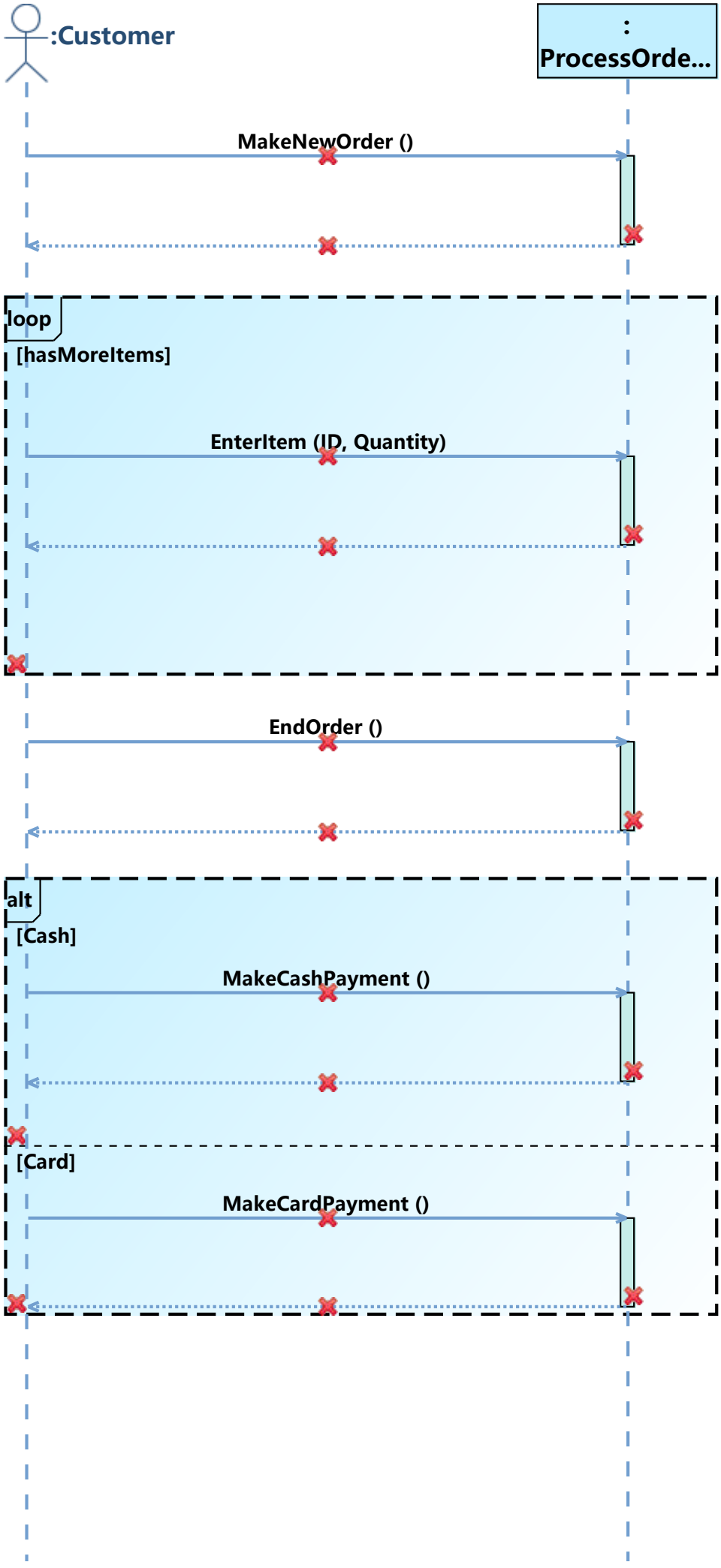
UC9 - manageCustomer

Use Case Description:

UseCase Name:	manageCustomer
UseCase ID:	UC9
Brief Description:	
Involved Actor:	Administer
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC10 - processOrder

System Sequence Diagram:


:Customer
:
ProcessOrde...
MakeNewOrder ()
loop
[hasMoreItems]
EnterItem (ID, Quantity)
EndOrder ()
alt
[Cash]
MakeCashPayment ()
[Card]
MakeCardPayment ()

Use Case Description:

UseCase Name:	processOrder
UseCase ID:	UC10
Brief Description:	
Involved Actor:	Customer
Preconditions:	
Postconditions:	
Basic Path:	1. Customer clicks to execute the operation makeNewOrder 2. Customer clicks to execute the operation enterItem, with entering id, quantity *If hasMoreItems, repeat the step(s) 2* 3. Customer clicks to execute the operation endOrder 4. Execute combinedFragement2 Select cash: Customer clicks to execute the operation makeCashPayment, with entering amount Select card: Customer clicks to execute the operation makeCardPayment
Alternative Path:	

UC11 - manageItem

Use Case Description:

UseCase Name:	manageItem
UseCase ID:	UC11
Brief Description:	
Involved Actor:	Business
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC12 - enterStore

Use Case Description:

UseCase Name:	enterStore
UseCase ID:	UC12
Brief Description:	
Involved Actor:	CustomerBusiness
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

UC13 - terminateOrder

Use Case Description:

UseCase Name:	terminateOrder
UseCase ID:	UC13
Brief Description:	
Involved Actor:	Dilivery
Preconditions:	
Postconditions:	
Basic Path:	
Alternative Path:	

3.1.2 System Operation

OP1 - createDilivery

Operation Name:	createDilivery
Operation ID:	OP1
Description:	
Service:	ManageDiliveryCRUDService
Input:	1. name: *id*, type: String 2. name: *name*, type: String
Output Type:	Boolean
Definition:	*di* is the object *ite* in the instance set of class Dilivery. *ite* represents an object of class Dilivery, and *ite* meets: The attribute *Id* of the object *ite* is equal to *id*
Preconditions:	The object *di* doesn't exist
Postconditions:	1. *temp* represented the object of class Dilivery 2. The object *temp* was created 3. The attribute *Id* of the object *temp* became *id* 4. The attribute *Name* of the object *temp* became *name* 5. The object *temp* was put into the instance set of class Dilivery 6. ERROR12 7. The return value was **true**

Contract of **createDilivery**:

```
Contract  ManageDiliveryCRUDService::createDilivery(id : String, name : String)
: Boolean {
        definition:
            di:Dilivery = Dilivery.allInstance()->any(ite:Dilivery | ite.Id =
id)
        precondition:
            di.oclIsUndefined() = true
        postcondition:
            let temp:Dilivery in
            temp.oclIsNew() and
            temp.Id = id and
            temp.Name = name and
            Dilivery.allInstance()->includes(temp) and
            CurrentDilivery = temp and
            result = true
}
```

OP2 - acceptOrder

Operation Name:	acceptOrder
Operation ID:	OP2
Description:	
Service:	TakeoutSystem
Input:	name: *name*, type: String
Output Type:	Boolean
Definition:	*order* is the object *s* in the instance set of class Sale. *s* represents an object of class Sale, and *s* meets: The attribute *Name* of the object *s* is equal to *name*
Preconditions:	1. The object *order* exists 2. The attribute *IsAccept* of the object *order* is equal to **false**
Postconditions:	1. The attribute *IsAccept* of the object *order* became **true** 2. The object *order* was linked to the object *CurrentDilivery* by *SaletoDilivery* 3. The object *CurrentDilivery* was linked to the object *order* by *DiliverytoSale* 4. The return value was **true**

Contract of **acceptOrder**:

```
Contract TakeoutSystem::acceptOrder(name : String) : Boolean {
        definition:
            order:Sale = Sale.allInstance()->any(s:Sale | s.Name = name)
        precondition:
            order.oclIsUndefined() = false and
            order.IsAccept = false
        postcondition:
            order.IsAccept = true and
            order.SaletoDilivery = CurrentDilivery and
            CurrentDilivery.DiliverytoSale->includes(order) and
            result = true
}
```

OP3 - terminateOrder

Operation Name:	terminateOrder
Operation ID:	OP3
Description:	
Service:	TakeoutSystem
Input:	name: *name*, type: String
Output Type:	Boolean
Definition:	*order* is the object *s* in the instance set of class Sale. *s* represents an object of class Sale, and *s* meets: The attribute *Name* of the object *s* is equal to *name*
Preconditions:	1. The object *order* exists 2. The attribute *IsAccept* of the object *order* is equal to **true** 3. The object *order* is linked to the object *CurrentDilivery* by *SaletoDilivery*
Postconditions:	1. The attribute *IsComplete* of the object *order* became **true** 2. The return value was **true**

Contract of **terminateOrder**:

```
Contract TakeoutSystem::terminateOrder(name : String) : Boolean {
        definition:
            order:Sale = Sale.allInstance()->any(s:Sale | s.Name = name)
        precondition:
            order.oclIsUndefined() = false and
            order.IsAccept = true and
            order.SaletoDilivery = CurrentDilivery
        postcondition:
            order.IsComplete = true and
            result = true
}
```

OP4 - excursionPublicOrder

Operation Name:	excursionPublicOrder
Operation ID:	OP4
Description:	
Service:	TakeoutSystem
Input:	name: *id*, type: String
Output Type:	Set of Sale
Definition:	*di* is the object *s* in the instance set of class Dilivery. *s* represents an object of class Dilivery, and *s* meets: The attribute *Id* of the object *s* is equal to *id*
Preconditions:	The object *di* exists
Postconditions:	The return value was the instance set of class Sale

Contract of **excursionPublicOrder**:

```
Contract TakeoutSystem::excursionPublicOrder(id : String) : Set(Sale) {
        /*
         * Generated by RM2DOc - Definition
         * cd is the object s in the instance set of class CashDesk. s
represents an object of class CashDesk, and s meets:
         *     The attribute Id of the object s is equal to cashDeskID
         */
        definition:
            di:Dilivery = Dilivery.allInstance()->any(s:Dilivery | s.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * cd exists
         * The attribute IsOpened of the object cd is equal to false
         * CurrentStore exists
         * The attribute IsOpened of the object CurrentStore is equal to true
         */
        precondition:
            di.oclIsUndefined() = false
        /*
         * Generated by RM2DOc - Postcondition
         * The object CurrentCashDesk became cd
         * The attribute IsOpened of the object cd became true
         * The return value was true
         */
        postcondition:
            result = Sale.allInstance()
}
```

OP5 - enterStore

Operation Name:	enterStore
Operation ID:	OP5
Description:	
Service:	TakeoutSystem
Input:	name: *id*, type: Integer
Output Type:	Boolean
Definition:	*store* is the object *s* in the instance set of class Store. *s* represents an object of class Store, and *s* meets: The attribute *Id* of the object *s* is equal to *id*
Preconditions:	The object *store* exists
Postconditions:	1. The object CurrentStore became *store* 2. The return value was **true**

Contract of **enterStore**:

```
Contract TakeoutSystem::enterStore(id : Integer) : Boolean {
        /*
         * Generated by RM2DOc - Definition
         * cd is the object s in the instance set of class CashDesk. s
represents an object of class CashDesk, and s meets:
         *     The attribute Id of the object s is equal to cashDeskID
         */
        definition:
            store:Store = Store.allInstance()->any(s:Store | s.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * cd exists
         * The attribute IsOpened of the object cd is equal to false
         * CurrentStore exists
         * The attribute IsOpened of the object CurrentStore is equal to true
         */
        precondition:
            store.oclIsUndefined() = false
        /*
         * Generated by RM2DOc - Postcondition
         * The object CurrentCashDesk became cd
         * The attribute IsOpened of the object cd became true
         * The return value was true
         */
        postcondition:
            self.CurrentStore = store and
            result = true
}
```

OP6 - createItem

Operation Name:	createItem
Operation ID:	OP6
Description:	
Service:	ManageItemCRUDService
Input:	1. name: *id*, type: Integer 2. name: *name*, type: String 3. name: *price*, type: Real 4. name: *stocknumber*, type: Integer 5. name: *orderprice*, type: Real
Output Type:	Boolean
Definition:	*item* is the object *ite* in the instance set of class Item. *ite* represents an object of class Item, and *ite* meets: The attribute *Id* of the object *ite* is equal to *id*
Preconditions:	1. The object *item* doesn't exist 2. The object *CurrentStore* exists
Postconditions:	1. *ite* represented the object of class Item 2. The object *ite* was created 3. The attribute *Id* of the object *ite* became *id* 4. The attribute *Name* of the object *ite* became *name* 5. The attribute *Price* of the object *ite* became *price* 6. The attribute *StockNumber* of the object *ite* became *stocknumber* 7. The attribute *OrderPrice* of the object *ite* became *orderprice* 8. The object *ite* was linked to the object *CurrentStore* by *ItemtoStore* 9. The object *ite* was put into the instance set of class Item 10. The return value was **true**

Contract of **createItem**:

```
Contract  ManageItemCRUDService::createItem(id : Integer, name : String, price : Real, stocknumber : Integer, orderprice : Real) : Boolean {
        /*
```

```
         * Generated by RM2DOc - Definition
         * item is the object ite in the instance set of class Item. ite
represents an object of class Item, and ite meets:
         *     The attribute Barcode of the object ite is equal to barcode
         */
        definition:
            item:Item = Item.allInstance()->any(ite:Item | ite.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * item doesn't exist
         */
        precondition:
            item.oclIsUndefined() = true and
            CurrentStore.oclIsUndefined() = false
        /*
         * Generated by RM2DOc - Postcondition
         * ite represented the object of class Item
         * The object ite was created
         * The attribute Barcode of the object ite became barcode
         * The attribute Name of the object ite became name
         * The attribute Price of the object ite became price
         * The attribute StockNumber of the object ite became stocknumber
         * The attribute OrderPrice of the object ite became orderprice
         * The object ite was put into the instance set of class Item
         * The return value was true
         */
        postcondition:
            let ite:Item in
            ite.oclIsNew() and
            ite.Id = id and
            ite.Name = name and
            ite.Price = price and
            ite.StockNumber = stocknumber and
            ite.OrderPrice = orderprice and
            ite.ItemtoStore = CurrentStore and
            Item.allInstance()->includes(ite) and
            result = true
}
```

OP7 - queryItem

Operation Name:	queryItem
Operation ID:	OP7
Description:	
Service:	ManageItemCRUDService
Input:	name: *id*, type: Integer
Output Type:	Item
Definition:	*item* is the object *ite* in the instance set of class Item. *ite* represents an object of class Item, and *ite* meets: The attribute *Id* of the object *ite* is equal to *id*
Preconditions:	The object *item* exists
Postconditions:	The return value was *item*

Contract of **queryItem**:

```
Contract  ManageItemCRUDService::queryItem(id : Integer) : Item {
        /*
         * Generated by RM2DOc - Definition
         * item is the object ite in the instance set of class Item. ite
represents an object of class Item, and ite meets:
         *     The attribute Barcode of the object ite is equal to barcode
         */
        definition:
            item:Item = Item.allInstance()->any(ite:Item | ite.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * item exists
         */
        precondition:
            item.oclIsUndefined() = false
        /*
         * Generated by RM2DOc - Postcondition
         * The return value was item
         */
        postcondition:
            result = item
}
```

OP8 - modifyItem

Operation Name:	modifyItem
Operation ID:	OP8
Description:	
Service:	ManageItemCRUDService
Input:	1. name: *id*, type: Integer 2. name: *name*, type: String 3. name: *price*, type: Real 4. name: *stocknumber*, type: Integer 5. name: *orderprice*, type: Real
Output Type:	Boolean
Definition:	*item* is the object *ite* in the instance set of class Item. *ite* represents an object of class Item, and *ite* meets: The attribute *Id* of the object *ite* is equal to *id*
Preconditions:	The object *item* exists
Postconditions:	1. The attribute *Id* of the object *item* became *id* 2. The attribute *Name* of the object *item* became *name* 3. The attribute *Price* of the object *item* became *price* 4. The attribute *StockNumber* of the object *item* became *stocknumber* 5. The attribute *OrderPrice* of the object *item* became *orderprice* 6. The return value was **true**

Contract of **modifyItem**:

```
Contract  ManageItemCRUDService::modifyItem(id : Integer, name : String, price :
Real, stocknumber : Integer, orderprice : Real) : Boolean {
        /*
         * Generated by RM2DOc - Definition
         * item is the object ite in the instance set of class Item. ite
represents an object of class Item, and ite meets:
         *     The attribute Barcode of the object ite is equal to barcode
         */
        definition:
            item:Item = Item.allInstance()->any(ite:Item | ite.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * item exists
         */
```

```
        precondition:
            item.oclIsUndefined() = false
        /*
         * Generated by RM2DOc - Postcondition
         * The attribute Barcode of the object item became barcode
         * The attribute Name of the object item became name
         * The attribute Price of the object item became price
         * The attribute StockNumber of the object item became stocknumber
         * The attribute OrderPrice of the object item became orderprice
         * The return value was true
         */
        postcondition:
            item.Id = id and
            item.Name = name and
            item.Price = price and
            item.StockNumber = stocknumber and
            item.OrderPrice = orderprice and
            result = true
}
```

OP9 - deleteItem

Operation Name:	deleteItem
Operation ID:	OP9
Description:	
Service:	ManageItemCRUDService
Input:	name: *id*, type: Integer
Output Type:	Boolean
Definition:	*item* is the object *ite* in the instance set of class Item. *ite* represents an object of class Item, and *ite* meets: The attribute *Id* of the object *ite* is equal to *id*
Preconditions:	1. The object *item* exists 2. The object *item* is in the instance set of class Item
Postconditions:	1. The object *item* was deleted from the instance set of class Item 2. The return value was **true**

Contract of **deleteItem**:

```
Contract  ManageItemCRUDService::deleteItem(id : Integer) : Boolean {
        /*
         * Generated by RM2DOc - Definition
```

```
         * item is the object ite in the instance set of class Item. ite
represents an object of class Item, and ite meets:
         *     The attribute Barcode of the object ite is equal to barcode
         */
        definition:
            item:Item = Item.allInstance()->any(ite:Item | ite.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * item exists
         * The object item is in the instance set of class Item
         */
        precondition:
            item.oclIsUndefined() = false and
            Item.allInstance()->includes(item)
        /*
         * Generated by RM2DOc - Postcondition
         * The object item was deleted from the instance set of class Item
         * The return value was true
         */
        postcondition:
            Item.allInstance()->excludes(item) and
            result = true
}
```

OP10 - search

Operation Name:	search
Operation ID:	OP10
Description:	
Service:	TakeoutSystem
Input:	name: *itemname*, type: String
Output Type:	Item
Definition:	*item* is the object *ite* in the instance set of class Item. *ite* represents an object of class Item, and *ite* meets: The attribute *Name* of the object *ite* is equal to *itemname* The object *ite* is linked to the object *CurrentStore* by *ItemtoStore*
Preconditions:	The object *item* exists
Postconditions:	The return value was *item*

Contract of **search**:

```
Contract TakeoutSystem::search(itemname : String) : Item {
        /*
```

```
         * Generated by RM2DOc - Definition
         * item is the object ite in the instance set of class Item. ite
represents an object of class Item, and ite meets:
         *     The attribute Barcode of the object ite is equal to barcode
         */
        definition:
            item:Item = Item.allInstance()->any(ite:Item | ite.Name = itemname
and ite.ItemtoStore = CurrentStore)
        /*
         * Generated by RM2DOc - Precondition
         * item exists
         */
        precondition:
            item.oclIsUndefined() = false
        /*
         * Generated by RM2DOc - Postcondition
         * The return value was item
         */
        postcondition:
            result = item
}
```

OP11 - createStore

Operation Name:	createStore
Operation ID:	OP11
Description:	
Service:	ManageStoreCRUDService
Input:	1. name: *id*, type: Integer 2. name: *name*, type: String 3. name: *address*, type: String 4. name: *isopened*, type: Boolean
Output Type:	Boolean
Definition:	*store* is the object *sto* in the instance set of class Store. *sto* represents an object of class Store, and *sto* meets: The attribute *Id* of the object *sto* is equal to *id*
Preconditions:	The object *store* doesn't exist
Postconditions:	1. *sto* represented the object of class Store 2. The object *sto* was created 3. The attribute *Id* of the object *sto* became *id* 4. The attribute *Name* of the object *sto* became *name* 5. The attribute *Address* of the object *sto* became *address* 6. The attribute *IsOpened* of the object *sto* became *isopened* 7. The object *sto* was put into the instance set of class Store 8. The return value was **true**

Contract of **createStore**:

```
Contract  ManageStoreCRUDService::createStore(id : Integer, name : String,
address : String, isopened : Boolean) : Boolean {
        /*
         * Generated by RM2DOc - Definition
         * store is the object sto in the instance set of class Store. sto
represents an object of class Store, and sto meets:
         *     The attribute Id of the object sto is equal to id
         */
        definition:
            store:Store = Store.allInstance()->any(sto:Store | sto.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * store doesn't exist
```

```
         */
        precondition:
            store.oclIsUndefined() = true
        /*
         * Generated by RM2DOc - Postcondition
         * sto represented the object of class Store
         * The object sto was created
         * The attribute Id of the object sto became id
         * The attribute Name of the object sto became name
         * The attribute Address of the object sto became address
         * The attribute IsOpened of the object sto became isopened
         * The object sto was put into the instance set of class Store
         * The return value was true
         */
        postcondition:
            let sto:Store in
            sto.oclIsNew() and
            sto.Id = id and
            sto.Name = name and
            sto.Address = address and
            sto.IsOpened = isopened and
            Store.allInstance()->includes(sto) and
            result = true
}
```

OP12 - queryStore

Operation Name:	queryStore
Operation ID:	OP12
Description:	
Service:	ManageStoreCRUDService
Input:	name: *id*, type: Integer
Output Type:	Store
Definition:	*store* is the object *sto* in the instance set of class Store. *sto* represents an object of class Store, and *sto* meets: The attribute *Id* of the object *sto* is equal to *id*
Preconditions:	The object *store* exists
Postconditions:	The return value was *store*

Contract of **queryStore**:

```
Contract  ManageStoreCRUDService::queryStore(id : Integer) : Store {
        /*
         * Generated by RM2DOc - Definition
```

```
         * store is the object sto in the instance set of class Store. sto
represents an object of class Store, and sto meets:
         *     The attribute Id of the object sto is equal to id
         */
        definition:
            store:Store = Store.allInstance()->any(sto:Store | sto.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * store exists
         */
        precondition:
            store.oclIsUndefined() = false
        /*
         * Generated by RM2DOc - Postcondition
         * The return value was store
         */
        postcondition:
            result = store
}
```

OP13 - modifyStore

Operation Name:	modifyStore
Operation ID:	OP13
Description:	
Service:	ManageStoreCRUDService
Input:	1. name: *id*, type: Integer 2. name: *name*, type: String 3. name: *address*, type: String 4. name: *isopened*, type: Boolean
Output Type:	Boolean
Definition:	*store* is the object *sto* in the instance set of class Store. *sto* represents an object of class Store, and *sto* meets: The attribute *Id* of the object *sto* is equal to *id*
Preconditions:	The object *store* exists
Postconditions:	1. The attribute *Id* of the object *store* became *id* 2. The attribute *Name* of the object *store* became *name* 3. The attribute *Address* of the object *store* became *address* 4. The attribute *IsOpened* of the object *store* became *isopened* 5. The return value was **true**

Contract of **modifyStore**:

```
Contract  ManageStoreCRUDService::modifyStore(id : Integer, name : String,
address : String, isopened : Boolean) : Boolean {
        /*
         * Generated by RM2DOc - Definition
         * store is the object sto in the instance set of class Store. sto
represents an object of class Store, and sto meets:
         *     The attribute Id of the object sto is equal to id
         */
        definition:
            store:Store = Store.allInstance()->any(sto:Store | sto.Id = id)
        /*
         * Generated by RM2DOc - Precondition
         * store exists
         */
        precondition:
            store.oclIsUndefined() = false
        /*
         * Generated by RM2DOc - Postcondition
```

```
         * The attribute Id of the object store became id
         * The attribute Name of the object store became name
         * The attribute Address of the object store became address
         * The attribute IsOpened of the object store became isopened
         * The return value was true
         */
        postcondition:
            store.Id = id and
            store.Name = name and
            store.Address = address and
            store.IsOpened = isopened and
            result = true
}
```

OP14 - deleteStore

Operation Name:	deleteStore
Operation ID:	OP14
Description:	
Service:	ManageStoreCRUDService
Input:	name: *id*, type: Integer
Output Type:	Boolean
Definition:	*store* is the object *sto* in the instance set of class Store. *sto* represents an object of class Store, and *sto* meets: The attribute *Id* of the object *sto* is equal to *id*
Preconditions:	1. The object *store* exists 2. The object *store* is in the instance set of class Store
Postconditions:	1. The object *store* was deleted from the instance set of class Store 2. The return value was **true**

Contract of **deleteStore**:

```
Contract  ManageStoreCRUDService::deleteStore(id : Integer) : Boolean {
        /*
         * Generated by RM2DOc - Definition
         * store is the object sto in the instance set of class Store. sto
represents an object of class Store, and sto meets:
         *     The attribute Id of the object sto is equal to id
         */
        definition:
            store:Store = Store.allInstance()->any(sto:Store | sto.Id = id)
        /*
```

```
         * Generated by RM2DOc - Precondition
         * store exists
         * The object store is in the instance set of class Store
         */
        precondition:
            store.oclIsUndefined() = false and
            Store.allInstance()->includes(store)
        /*
         * Generated by RM2DOc - Postcondition
         * The object store was deleted from the instance set of class Store
         * The return value was true
         */
        postcondition:
            Store.allInstance()->excludes(store) and
            result = true
}
```

OP15 - makeNewOrder

Operation Name:	makeNewOrder
Operation ID:	OP15
Description:	
Service:	ProcessOrderService
Input:	None
Output Type:	Boolean
Preconditions:	1. The object *CurrentStore* exists 2. (the object *CurrentSale* doesn't exist, or (the object *CurrentSale* exists, and the attribute *IsComplete* of the object *CurrentSale* is equal to **true**))
Postconditions:	1. *s* represented the object of class Sale 2. The object *s* was created 3. The object *s* was linked to the object *CurrentStore* by *SaletoStore* 4. The object *CurrentStore* was linked to the object *s* by *StoretoSale* 5. The attribute *IsComplete* of the object *s* became **false** 6. The attribute *IsReadytoPay* of the object *s* became **false** 7. The object *s* was put into the instance set of class Sale 8. The object CurrentSale became *s* 9. The return value was **true**

Contract of **makeNewOrder**:

```
Contract ProcessOrderService::makeNewOrder() : Boolean {
```

```
        /*
         * Generated by RM2DOc - Precondition
         * CurrentCashDesk exists
         * The attribute IsOpened of the object CurrentCashDesk is equal to true
         * (CurrentSale doesn't exist, or (CurrentSale exists, and the attribute
IsComplete of the object CurrentSale is equal to true))
         */
        precondition:
            CurrentStore.oclIsUndefined() = false and
            (CurrentSale.oclIsUndefined() = true or
                (CurrentSale.oclIsUndefined() = false and
                    CurrentSale.IsComplete = true
                )
            )
        /*
         * Generated by RM2DOc - Postcondition
         * s represented the object of class Sale
         * The object s was created
         * The object s was linked to the object CurrentCashDesk by
BelongedCashDesk
         * The object CurrentCashDesk was linked to the object s by
ContainedSales
         * The attribute IsComplete of the object s became false
         * The attribute IsReadytoPay of the object s became false
         * The object s was put into the instance set of class Sale
         * The object CurrentSale became s
         * The return value was true
         */
        postcondition:
            let s:Sale in
            s.oclIsNew() and
            s.SaletoStore = CurrentStore and
            CurrentStore.StoretoSale->includes(s) and
            s.IsComplete = false and
            s.IsReadytoPay = false and
            Sale.allInstance()->includes(s) and
            self.CurrentSale = s and
            result = true
}
```

OP16 - enterItem

Operation Name:	enterItem
Operation ID:	OP16
Description:	
Service:	ProcessOrderService
Input:	1. name: *id*, 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 *Id* of the object *i* is equal to *id*
Preconditions:	1. The object *CurrentSale* exists 2. The attribute *IsComplete* of the object *CurrentSale* is equal to **false** 3. The object *item* exists 4. The attribute *StockNumber* of the object *item* is greater than **0**
Postconditions:	1. *sli* represented the object of class OrderLineItem 2. The object *sli* was created 3. The object CurrentOrderLine became *sli* 4. The object *sli* was linked to the object *CurrentSale* by *OrderLineItemtoSale* 5. The object *CurrentSale* was linked to the object *sli* by *SaletoOrderLineItem* 6. The attribute *Quantity* of the object *sli* became *quantity* 7. The object *sli* was linked to the object *item* by *OrderLineItemtoItem* 8. The attribute *StockNumber* of the object *item* became its previous value 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 OrderLineItem 11. The return value was **true**

Contract of **enterItem**:

```
Contract ProcessOrderService::enterItem(id : 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.Id = id)
        /*
         * 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:OrderLineItem in
            sli.oclIsNew() and
            self.CurrentOrderLine = sli and
            sli.OrderLineItemtoSale = CurrentSale and
            CurrentSale.SaletoOrderLineItem->includes(sli) and
            sli.Quantity = quantity and
            sli.OrderLineItemtoItem = item and
            item.StockNumber = item.StockNumber@pre - quantity and
            sli.Subamount = item.Price * quantity and
            OrderLineItem.allInstance()->includes(sli) and
            result = true
}
```

OP17 - endOrder

Operation Name:	endOrder
Operation ID:	OP17
Description:	
Service:	ProcessOrderService
Input:	None
Output Type:	Real
Definition:	1. *sls* is the Set of class OrderLineItem, including which *CurrentSale* is linked to 2. *sub* is the Set of Real, including the *Subamount* of each object in the set *sls*
Preconditions:	1. The object *CurrentSale* exists 2. The attribute *IsComplete* of the object *CurrentSale* is equal to **false** 3. The attribute *IsReadytoPay* of the object *CurrentSale* is equal to **false**
Postconditions:	1. The attribute *Amount* of the object *CurrentSale* became the sum of *sub* 2. The attribute *IsReadytoPay* of the object *CurrentSale* became **true** 3. The return value was the attribute *Amount* of the object *CurrentSale*

Contract of **endOrder**:

```
Contract ProcessOrderService::endOrder() : Real {
        /*
         * Generated by RM2DOc - Definition
         * sls is the Set of class SalesLineItem, including  which CurrentSale
is linked to
         * sub is the Set of Real, including the Subamount of each object in the
set sls
         */
        definition:
            sls:Set(OrderLineItem) = CurrentSale.SaletoOrderLineItem,
            sub:Set(Real) = sls->collect(s:OrderLineItem | s.Subamount)
        /*
         * Generated by RM2DOc - Precondition
         * CurrentSale exists
         * The attribute IsComplete of the object CurrentSale is equal to false
         * The attribute IsReadytoPay of the object CurrentSale is equal to
false
         */
        precondition:
            CurrentSale.oclIsUndefined() = false and
            CurrentSale.IsComplete = false and
            CurrentSale.IsReadytoPay = false
        /*
```

```
         * Generated by RM2DOc - Postcondition
         * The attribute Amount of the object CurrentSale became the sum of sub
         * The attribute IsReadytoPay of the object CurrentSale became true
         * The return value was the attribute Amount of the object CurrentSale
         */
        postcondition:
            CurrentSale.Amount = sub.sum() and
            CurrentSale.IsReadytoPay = true and
            result = CurrentSale.Amount
}
```

OP18 - makeCashPayment

Operation Name:	makeCashPayment
Operation ID:	OP18
Description:	
Service:	ProcessOrderService
Input:	name: *amount*, type: Real
Output Type:	Boolean
Preconditions:	1. The object *CurrentSale* exists 2. The attribute *IsComplete* of the object *CurrentSale* is equal to **false** 3. The attribute *IsReadytoPay* of the object *CurrentSale* is equal to **true** 4. The *amount* is greater than or equal to the attribute *Amount* of the object *CurrentSale*
Postconditions:	1. *cp* represented the object of class CashPayment 2. The object *cp* was created 3. The attribute *AmountTendered* of the object *cp* became *amount* 4. The object *cp* was linked to the object *CurrentSale* by *PaymenttoSale* 5. The object *CurrentSale* was linked to the object *cp* by *SaletoPayment* 6. The object *CurrentSale* was linked to the object *CurrentStore* by *SaletoStore* 7. The object *CurrentStore* was linked to the object *CurrentSale* by *StoretoSale* 8. The attribute *Balance* of the object *cp* became *amount* minus the attribute *Amount* of the object *CurrentSale* 9. The object *cp* was put into the instance set of class CashPayment 10. The attribute *IsAccept* of the object *CurrentSale* became **false** 11. The attribute *Name* of the object *CurrentSale* became the attribute *Name* of the object *CurrentStore* 12. The return value was **true**

Contract of **makeCashPayment**:

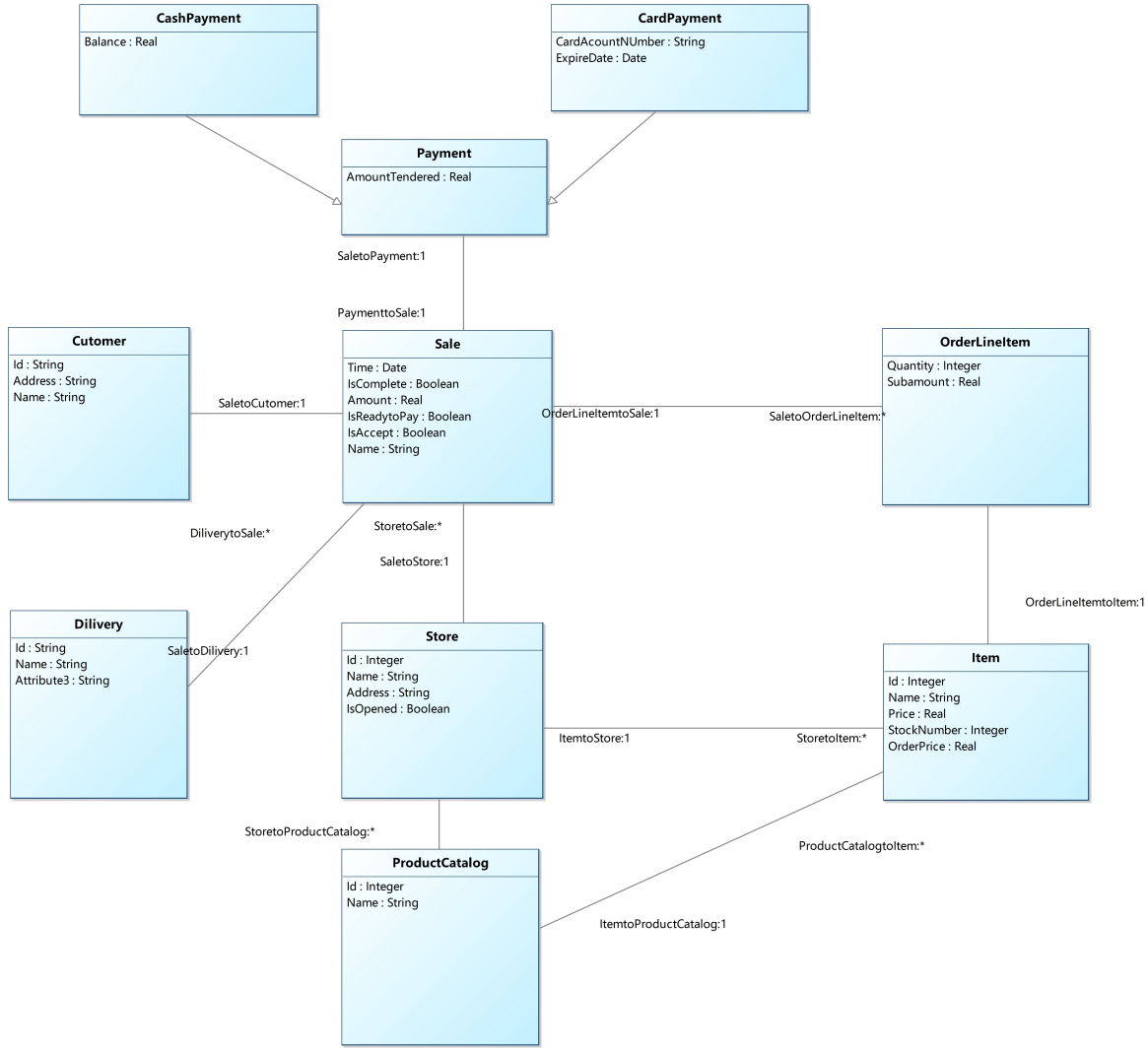
```
Contract ProcessOrderService::makeCashPayment(amount : Real) : Boolean {
        /*
         * Generated by RM2DOc - Precondition
         * CurrentSale exists
         * The attribute IsComplete of the object CurrentSale is equal to false
         * The attribute IsReadytoPay of the object CurrentSale is equal to true
```

```
         * The amount is greater than or equal to the attribute Amount of the
object CurrentSale
         */
        precondition:
            CurrentSale.oclIsUndefined() = false and
            CurrentSale.IsComplete = false and
            CurrentSale.IsReadytoPay = true and
            amount >= CurrentSale.Amount
        /*
         * Generated by RM2DOc - Postcondition
         * cp represented the object of class CashPayment
         * The object cp was created
         * The attribute AmountTendered of the object cp became amount
         * The object cp was linked to the object CurrentSale by BelongedSale
         * The object CurrentSale was linked to the object cp by
AssoicatedPayment
         * The object CurrentSale was linked to the object CurrentStore by
Belongedstore
         * The object CurrentStore was linked to the object CurrentSale by Sales
         * The attribute IsComplete of the object CurrentSale became true
         * The attribute Time of the object CurrentSale was equal to Now
         * The attribute Balance of the object cp became amount minus the
attribute Amount of the object CurrentSale
         * The object cp was put into the instance set of class CashPayment
         * The return value was true
         */
        postcondition:
            let cp:CashPayment in
            cp.oclIsNew() and
            cp.AmountTendered = amount and
            cp.PaymenttoSale = CurrentSale and
            CurrentSale.SaletoPayment = cp and
            CurrentSale.SaletoStore = CurrentStore and
            CurrentStore.StoretoSale->includes(CurrentSale) and
            cp.Balance = amount - CurrentSale.Amount and
            CashPayment.allInstance()->includes(cp) and
            CurrentSale.IsAccept = false and
            CurrentSale.Name = CurrentStore.Name and
            result = true
}
```

3.2 Database requirements

3.2.1 Entity Analysis

Conceptual Class Diagram



E1 - Item

Entity Name:	Item	
Entity ID:	E1	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	Integer	The Id 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**
ItemtoProductCatalog	ProductCatalog	Association: One-to-One
ItemtoStore	Store	Association: One-to-One

E2 - OrderLineItem

Entity Name:	OrderLineItem	
Entity ID:	E2	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Quantity	Integer	The Quantity of OrderLineItem
Subamount	Real	The Subamount of OrderLineItem
Relationship Name	**Related Entity**	**Relationship Type**
OrderLineItemtoItem	Item	Association: One-to-One
OrderLineItemtoSale	Sale	Association: One-to-One

E3 - Sale

Entity Name:	Sale	
Entity ID:	E3	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Time	LocalDate	The Time of Sale
IsComplete	Boolean	The IsComplete of Sale
Amount	Real	The Amount of Sale
IsReadytoPay	Boolean	The IsReadytoPay of Sale
IsAccept	Boolean	The IsAccept of Sale
Name	String	The Name of Sale
Relationship Name	**Related Entity**	**Relationship Type**
SaletoOrderLineItem	OrderLineItem	Association: One-to-Many
SaletoPayment	Payment	Association: One-to-One
SaletoStore	Store	Association: One-to-One
SaletoCutomer	Cutomer	Association: One-to-One
SaletoDilivery	Dilivery	Association: One-to-One

E4 - Payment

Entity Name:	Payment	
Entity ID:	E4	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
AmountTendered	Real	The AmountTendered of Payment
Relationship Name	**Related Entity**	**Relationship Type**
PaymenttoSale	Sale	Association: One-to-One

E5 - CashPayment

Entity Name:	CashPayment	
Entity ID:	E5	
Entity Description:		
Super Entity:	Payment	
Attribute Name	**Attribute Type**	**Attribute Description**
Balance	Real	The Balance of CashPayment

E6 - CardPayment

Entity Name:	CardPayment	
Entity ID:	E6	
Entity Description:		
Super Entity:	Payment	
Attribute Name	**Attribute Type**	**Attribute Description**
CardAcountNUmber	String	The CardAcountNUmber of CardPayment
ExpireDate	LocalDate	The ExpireDate of CardPayment

E7 - Store

Entity Name:	Store	
Entity ID:	E7	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	Integer	The Id of Store
Name	String	The Name of Store
Address	String	The Address of Store
IsOpened	Boolean	The IsOpened of Store
Relationship Name	**Related Entity**	**Relationship Type**
StoretoSale	Sale	Association: One-to-Many
StoretoItem	Item	Association: One-to-Many
StoretoProductCatalog	ProductCatalog	Association: One-to-Many

E8 - ProductCatalog

Entity Name:	ProductCatalog	
Entity ID:	E8	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	Integer	The Id of ProductCatalog
Name	String	The Name of ProductCatalog
Relationship Name	**Related Entity**	**Relationship Type**
ProductCatalogtoItem	Item	Association: One-to-Many

E9 - Cutomer

Entity Name:	Cutomer	
Entity ID:	E9	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	String	The Id of Cutomer
Address	String	The Address of Cutomer
Name	String	The Name of Cutomer

E10 - Dilivery

Entity Name:	Dilivery	
Entity ID:	E10	
Entity Description:		
Attribute Name	**Attribute Type**	**Attribute Description**
Id	String	The Id of Dilivery
Name	String	The Name of Dilivery
Attribute3	String	The Attribute3 of Dilivery
Relationship Name	**Related Entity**	**Relationship Type**
DiliverytoSale	Sale	Association: One-to-Many

3.2.2 Other database requirements

This should specify the logical requirements for any information that is to be placed into a database. This may include the following:

- a) Types of information used by various functions;
- b) Frequency of use;
- c) Accessing capabilities;
- d) Integrity constraints;
- e) Data retention requirements.

3.3 Performance requirements

3.3.1 Static numerical requirements

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole. Static numerical requirements may include the following:

- a) The number of terminals to be supported;
- b) The number of simultaneous users to be supported;
- c) Amount and type of information to be handled.

3.3.2 Dynamic numerical requirements

Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

All of these requirements should be stated in measurable terms.

For example,

- *95% of the transactions shall be processed in less than 1 s.*

rather than,

- *An operator shall not have to wait for the transaction to complete.*

NOTE:Numerical limits applied to one specifific function are normally specifified as part of the processing subparagraph description of that function.

3.4 Usability requirements

Define usability and quality in use requirements and objectives for the software system that can include measurable effectiveness, efficiency, satisfaction criteria and avoidance of harm that could arise from use in specific contexts of use.

3.5 Interface requirements

3.5.1 User interfaces

This should specify the following:

- a) The logical characteristics of each interface between the software product and its users. This includes those configuration characteristics (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.
- b) All the aspects of optimizing the interface with the person who must use the system. This may simply comprise a list of do's and don'ts on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be verifiable, e.g., "a clerk typist grade 4 can do function X in Z min after 1 h of training" rather than "a typist can do function X." (This may also be specified in the Software System Attributes under a section titled Ease of Use.)

3.5.2 Hardware interfaces

This should specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support.

3.5.3 Software interfaces

This should specify the use of other required software products (e.g., a data management system, an operating system, or a mathematical package), and interfaces with other application systems (e.g., the linkage between an accounts receivable system and a general ledger system). For each required software product, the following should be provided:

- a) Name;
- b) Mnemonic;
- c) Specification number;
- d) Version number;
- e) Source.

For each interface, the following should be provided:

- a) Discussion of the purpose of the interfacing software as related to this software product.
- b) Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

3.5.4 Communications interfaces

This should specify the various interfaces to communications such as local network protocols, etc.

3.6 Design constraints

Specify constraints on the system design imposed by external standards, regulatory requirements or project limitations.

3.6.1 Standards compliance

This subsection should specify the requirements derived from existing standards or regulations. They may include the following:

- a) Report format;
- b) Data naming;
- c) Accounting procedures;
- d) Audit tracing.

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

3.7 Software system attributes

3.7.1 Reliability

This should specify the factors required to establish the required reliability of the software system at time of delivery.

3.7.2 Availability

This should specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart.

3.7.3 Security

This should specify the factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to

- a) Utilize certain cryptographical techniques;
- b) Keep specific log or history data sets;
- c) Assign certain functions to different modules;
- d) Restrict communications between some areas of the program;
- e) Check data integrity for critical variables.

3.7.4 Maintainability

This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

3.7.5 Portability

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:

- a) Percentage of components with host-dependent code;
- b) Percentage of code that is host dependent;
- c) Use of a proven portable language;
- d) Use of a particular compiler or language subset;
- e) Use of a particular operating system.

3.8 Supporting information

Additional supporting information to be considered includes:

- a) sample input/output formats, descriptions of cost analysis studies or results of user surveys;
- b) supporting or background information that can help the readers of the SRS;
- c) a description of the problems to be solved by the software; and
- d) special packaging instructions for the code and the media to meet security, export, initial loading or other requirements.

The SRS should explicitly state whether or not these information items are to be considered part of the requirements.

4 Verification

Provide the verification approaches and methods planned to qualify the software. The information items for verification are recommended to be given in a parallel manner with the information items in Section 3.

5 Appendices

5.1 Assumptions and dependencies

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

5.2 Apportioning of requirements

Apportion the software requirements to software elements. For requirements that will require implementation over multiple software elements, or when allocation to a software element is initially undefined, this should be so stated. A cross-reference table by function and software element should be used to summarize the apportionments.

Identify requirements that may be delayed until future versions of the system (e.g., blocks and/or increments).

5.3 Acronyms and abbreviations

This subsection should provide the acronyms and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.