Accounts Receivable Processor (Design by Contract)

Software Engineering 2

Author: Sagar Singh
Student ID: C22731795
Program: TU856

Module: CMPU2020

30.04.2024

1 Table of Contents

1 Introduction

The aim of this report is to construct and test a USE model for Accounts Receivable Processor (Frankel, 2003) by Design by Contract. Design by Contract is an approach to designing software where specifications for software components should defined formally and explicitly. This includes implementation of pre-conditions and post-conditions for each operation. These become assertions for the operations. So, operations can only be executed as intended. Also, it uses invariants which are assertions that must be true throughout the whole process of the state, besides during an execution of an operation. (Frankel, 2003)

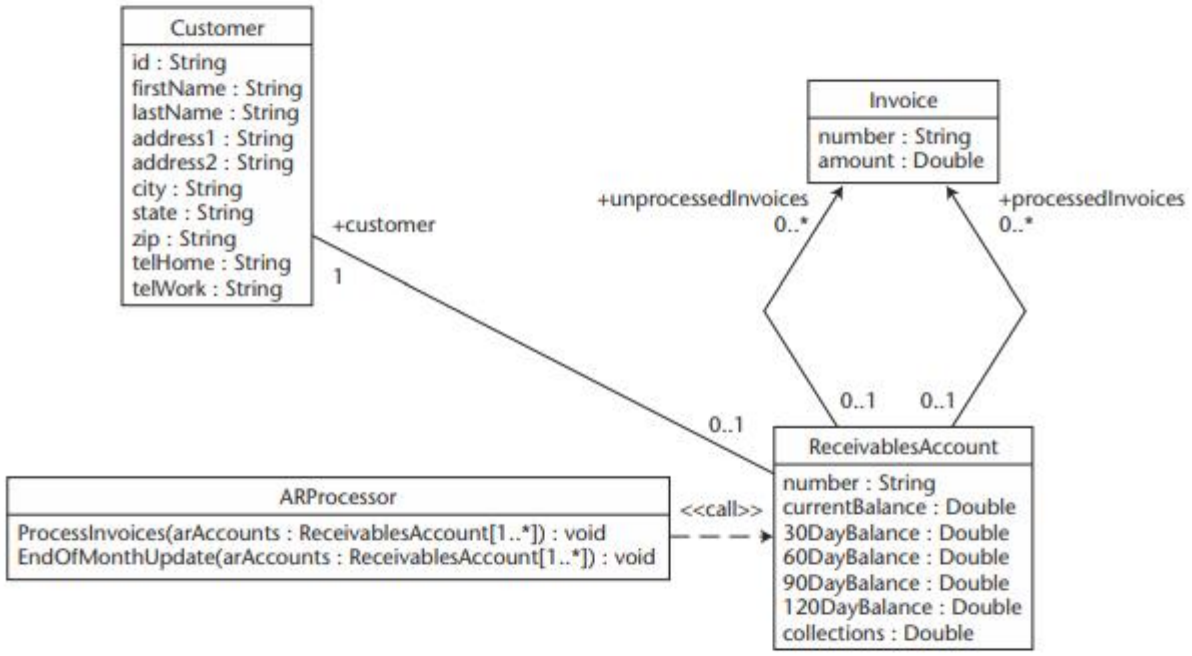
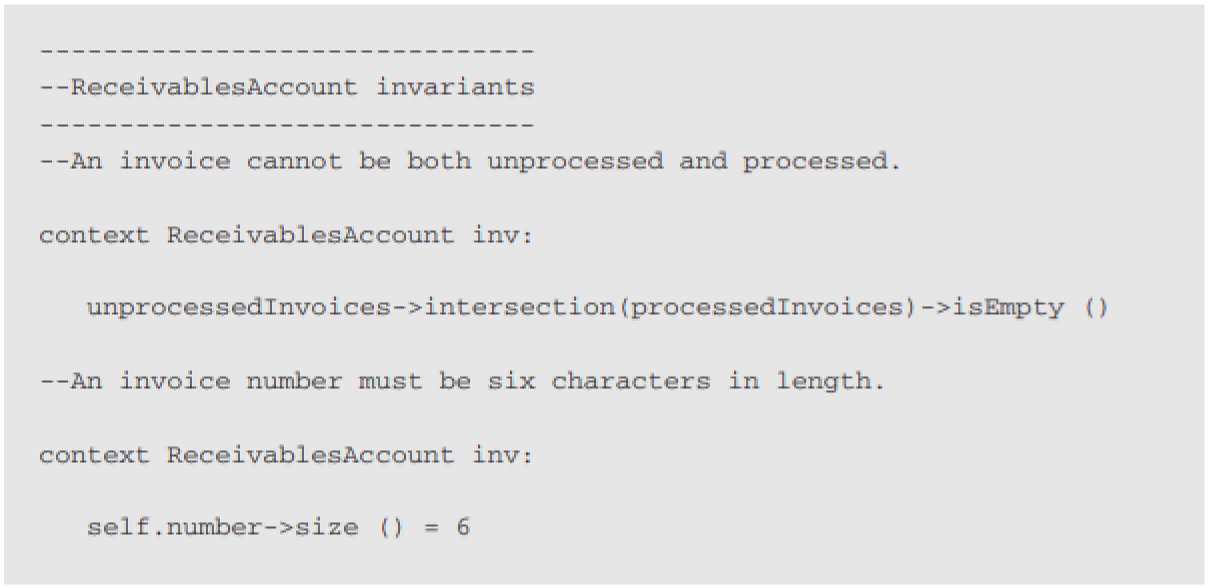
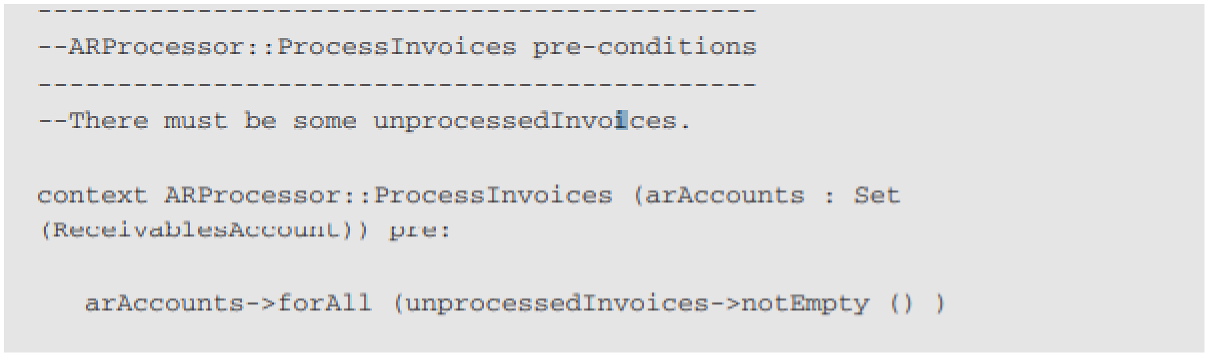


Figure 1. Simple UML class model from page 80. Frankel's Applying MDA to Enterprise Computing

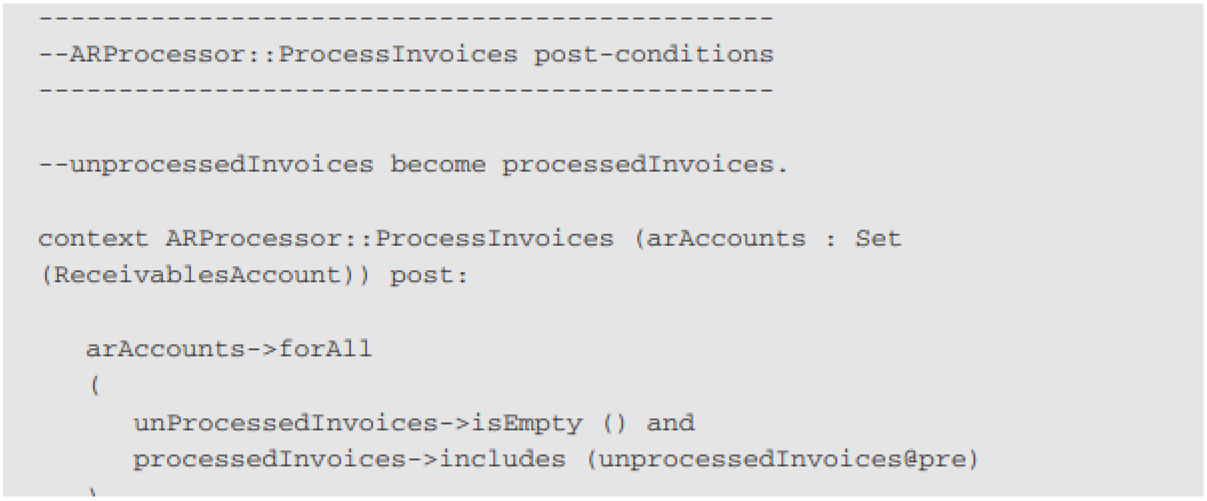
Initially I will attempt to replicate this model in USE. Then add the Design by Contract enhancements given to the model given below.



```
-------------------------------
--ReceivablesAccount invariants
-------------------------------
--An invoice cannot be both unprocessed and processed.

context ReceivablesAccount inv:

   unprocessedInvoices->intersection(processedInvoices)->isEmpty ()

--An invoice number must be six characters in length.

context ReceivablesAccount inv:

   self.number->size () = 6
```



```
---------------------------------------------
--ARProcessor::ProcessInvoices pre-conditions
---------------------------------------------
--There must be some unprocessedInvoices.

context ARProcessor::ProcessInvoices (arAccounts : Set
(ReceivablesAccount)) pre:

   arAccounts->forAll (unprocessedInvoices->notEmpty () )
```



```
----------------------------------------------
--ARProcessor::ProcessInvoices post-conditions
----------------------------------------------

--unprocessedInvoices become processedInvoices.

context ARProcessor::ProcessInvoices (arAccounts : Set
(ReceivablesAccount)) post:

   arAccounts->forAll
   (
      unProcessedInvoices->isEmpty () and
      processedInvoices->includes (unprocessedInvoices@pre)
   )
```

```
---------------------------------------------
--ARProcessor::EndOfMonthUpdate pre-conditions
---------------------------------------------
--There are no unprocessed invoices.

context ARProcessor::EndOfMonthUpdate (arAccounts : Set
(ReceivablesAccount)) pre:

   arAccounts->forAll (unprocessedInvoices->isEmpty () )
```

```
----------------------------------------------
--ARProcessor::EndOfMonthUpdate post-conditions
----------------------------------------------
--For all of the ARaccounts the following holds:
   --The Collections value is its previous value plus the previous
120DayBalance and
   --the 120DayBalance is the previous 90DayBalance and
   --the 90DayBalance is the previous 60DayBalance and
   --the 60DayBalance is the previous 30DayBalance and
   --the 30DayBalance is the previous currentBalance
   --the currentBalance is 0.

context ARProcessor::EndOfMonthUpdate (arAccounts : Set
(ReceivablesAccount)) post:

   arAccounts->forAll
   (
      -- @pre modifies an identifier to refer to the value it had
      -- before the operation executed.
      currentBalance = 0 and
      30DayBalance =  currentBalance@pre and
      60DayBalance =  30DayBalance@pre and
      90DayBalance =  60DayBalance@pre and
      120DayBalance = 90DayBalance@pre and
      Collections = collections@pre + 120DayBalance@pre
   )
```

Finally, I will adapt the model to add a few more constraints, adjust the operations, add state machines, and create a new operation creating an invoice. All the constraints and operations will be tested vigorously, and diagrams and command window outputs will be showcased.

2 Planning & Design

From figure 1. It can be established what each class's attributes and operations are. Associations are clearly described as well in the model. The task is to only construct this model in USE and a few parts to develop the model, therefore there is not much to design or plan.

Alterations:

1. New constraints
    a. Invariant that all invoices' number must have a uniquely identifiable number
    b. Pre-condition for ARProcessor's CreateInvoice() that the account passed must exist to add the invoice to
    c. Post-condition for ReceivablesAccount's NewInvoice(), called from CreateInvoice(), should make sure the new invoice has been added to the set of unprocessed invoices of the account
2. Operation adjustments
    a. Instead of the operations in ARProcessor, it calls the ReceivablesAccount respective operations for ProcessInvoices() and EndOfMonthUpdate(). Because the ARProcessor should only invoke the set of ReceivablesAccounts passed to do their processing since the processing does not require the accounts to interact with each other.
3. State machines
    a. Add State Machine to both ARProcessor and ReceivablesAccount that identically describes the current system and account state, which shows if the objects have any unprocessed invoices or not.
4. New operation creating an invoice
    a. From ARProcessor CreateInvoice() it takes the number and amount for the new the invoice being made, and the account it is being added to. After creation, ReceivablesAccount's NewInvoice() is called for the account while passing the new invoice created.

b. ReceivablesAccount NewInvoice() will take in the new invoice and insert into the unprocessed invoices set.

3 Implementation

3.1 Classes and associations

3.1.1 Class ReceivablesAccount

```
class ReceivablesAccount
    attributes
        number : String
        currentBalance : Real init = 0
        day30Balance : Real init = 0
        day60Balance : Real init = 0
        day90Balance : Real init = 0
        day120Balance : Real init = 0
        collections : Real init = 0
    operations
        ProcessInvoices()
        begin
            for invoice in self.unprocessedInvoices do
                    self.currentBalance := self.currentBalance + invoice.amount;
                    insert(self, invoice) into Processed;
                    delete(self, invoice) from Unprocessed;
            end
        end

        EndOfMonthUpdate()
        begin
                self.collections := self.collections + self.day120Balance;
                self.day120Balance := self.day90Balance;
                self.day90Balance := self.day60Balance;
                self.day60Balance := self.day30Balance;
                self.day30Balance := self.currentBalance;
                self.currentBalance := 0;
        end

        NewInvoice( invoice : Invoice )
        begin
            insert(self, invoice) into Unprocessed;
        end
    statemachines
        psm InvoicesProcessedStates
        states
            NewReceivablesAccount : initial
            UnprocessedState [self.unprocessedInvoices->size() > 0]
            ProcessedState [self.unprocessedInvoices->size() = 0]
        transitions
            NewReceivablesAccount -> ProcessedState { create }
            UnprocessedState -> ProcessedState { ProcessInvoices() }
            ProcessedState -> UnprocessedState { NewInvoice() }
            UnprocessedState -> UnprocessedState { NewInvoice() }
            ProcessedState -> ProcessedState { ProcessInvoices() }
        end
end
```

3.1.2 Class ARProcessor

```
class ARProcessor
    operations
        ProcessInvoices(arAccounts : Set(ReceivablesAccount))
        begin
            for recAcc in arAccounts do
                recAcc.ProcessInvoices();
            end
        end

        EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
        begin
            for recAcc in arAccounts do
                recAcc.EndOfMonthUpdate();
            end
        end

        CreateInvoice(number : String, amount : Real, acc : ReceivablesAccount)
        begin
            declare newInvoice : Invoice;
            newInvoice := new Invoice;
            newInvoice.number := number;
            newInvoice.amount := amount;
            acc.NewInvoice(newInvoice);
        end
    statemachines
```

```
        psm AllInvoicesProcessedStates
        states
            NewAPR : initial
            UnprocessedState
            ProcessedState
        transitions
            NewAPR -> ProcessedState { create }
            UnprocessedState -> ProcessedState { ProcessInvoices() }
            ProcessedState -> UnprocessedState { CreateInvoice() }
            UnprocessedState -> UnprocessedState { CreateInvoice() }
            ProcessedState -> ProcessedState { ProcessInvoices() }
        end
end
```

3.1.3 Class Customer

```
class Customer
    attributes
    id : String
    firstName : String
    lastName : String
    address : Address
    telHome : String
    telWork : String
end
```

3.1.4 Class Invoice

```
class Invoice
    attributes
        number : String
        amount : Real
end
```

3.1.5 Class Address

```
class Address
    attributes
        address1 : String
        address2 : String
        city : String
        state : String
        zip : String
end
```

3.1.6 Association between ReceivablesAccount and Customer

```
association Has between
    ReceivablesAccount[0..1]
    Customer[1] role customer
end
```

3.1.7 Association between ReceivablesAccount and Unprocessed Invoices

```
association Unprocessed between
    ReceivablesAccount[0..1] role unpRA
    Invoice[0..*] role unprocessedInvoices
end
```

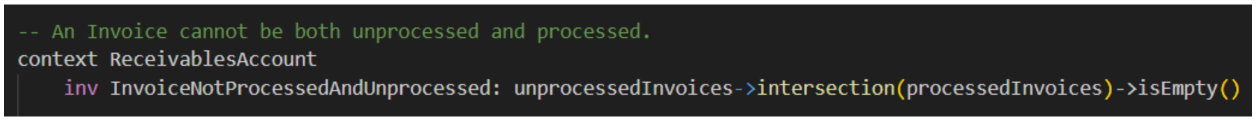
3.1.8 Association between ReceivablesAccount and Processed Invoices

```
association Processed between
    ReceivablesAccount[0..1] role pRA
    Invoice[0..*] role processedInvoices
end
```

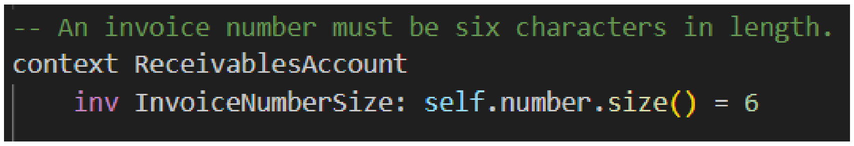
3.2 Constraints

3.2.1 Invariants

Applied to class ARProcessor:

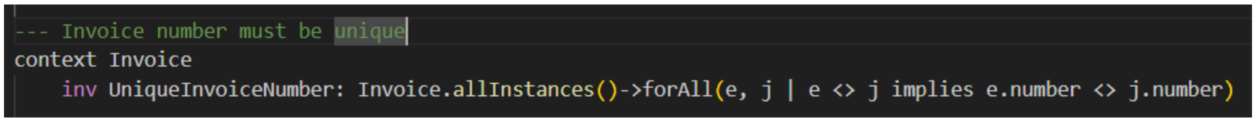
1. 

```
-- An Invoice cannot be both unprocessed and processed.
context ReceivablesAccount
    inv InvoiceNotProcessedAndUnprocessed: unprocessedInvoices->intersection(processedInvoices)->isEmpty()
```

2. 

```
-- An invoice number must be six characters in length.
context ReceivablesAccount
    inv InvoiceNumberSize: self.number.size() = 6
```

Applied to class Invoice:

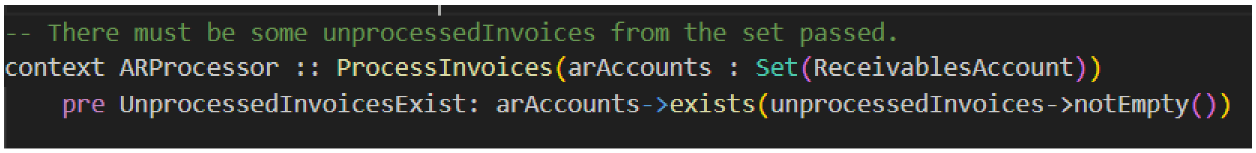
1. 

```
--- Invoice number must be unique
context Invoice
    inv UniqueInvoiceNumber: Invoice.allInstances()->forAll(e, j | e <> j implies e.number <> j.number)
```

3.2.2 Pre- & post-conditions

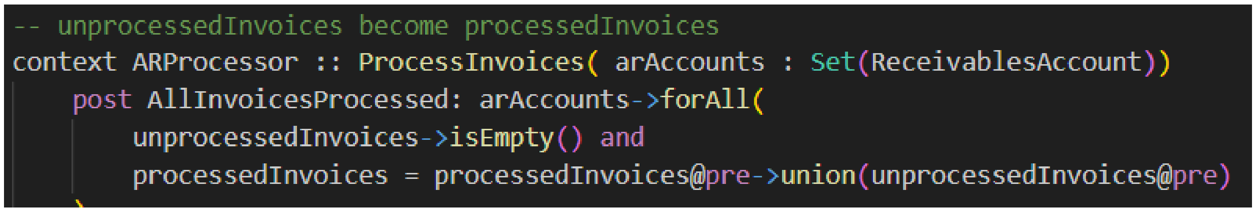
For ARProcessor's ProcessInvoices() operation:

1. Pre-condition:



```
-- There must be some unprocessedInvoices from the set passed.
context ARProcessor :: ProcessInvoices(arAccounts : Set(ReceivablesAccount))
    pre UnprocessedInvoicesExist: arAccounts->exists(unprocessedInvoices->notEmpty())
```

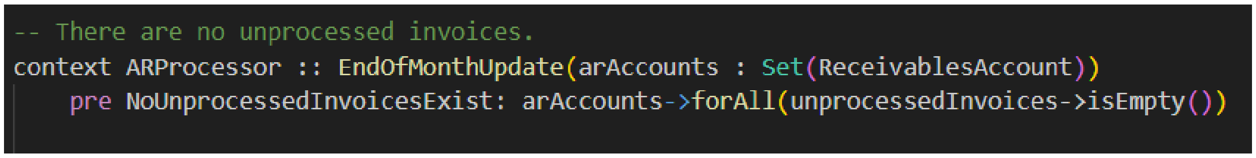
2. Post-condition:



```
-- unprocessedInvoices become processedInvoices
context ARProcessor :: ProcessInvoices( arAccounts : Set(ReceivablesAccount))
    post AllInvoicesProcessed: arAccounts->forAll(
        unprocessedInvoices->isEmpty() and
        processedInvoices = processedInvoices@pre->union(unprocessedInvoices@pre)
    )
```

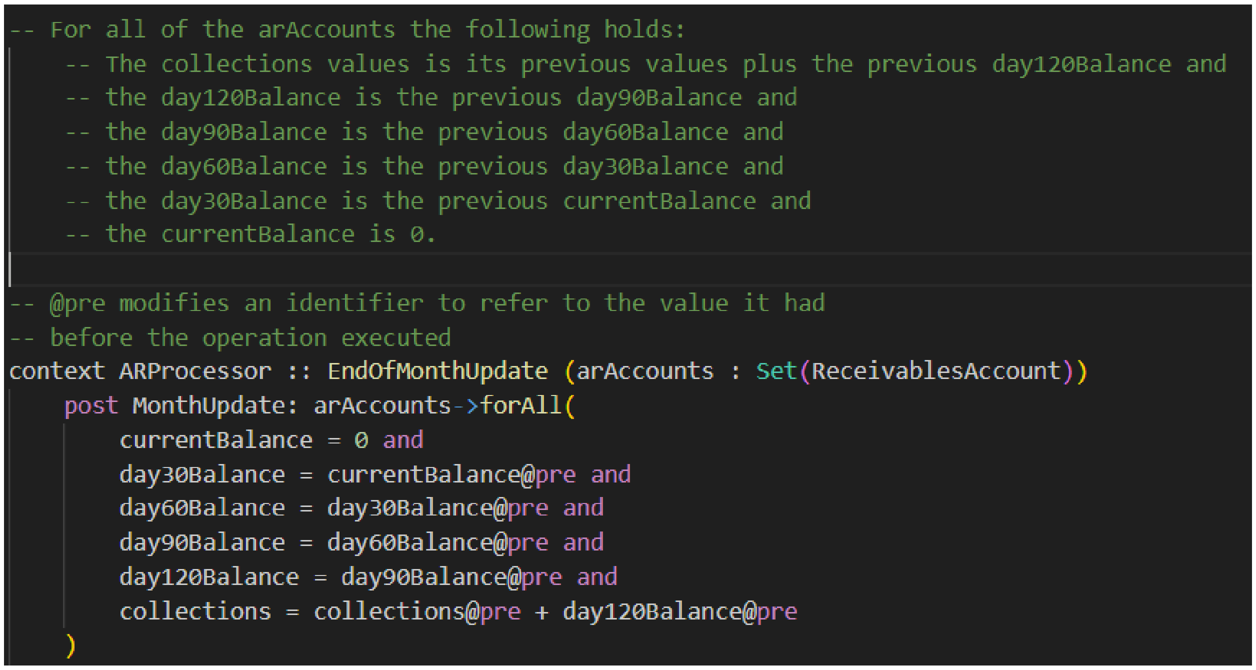
For ARProcessor’s EndOfMonthUpdate() operation:

1. Pre-condition:



```
-- There are no unprocessed invoices.
context ARProcessor :: EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
    pre NoUnprocessedInvoicesExist: arAccounts->forAll(unprocessedInvoices->isEmpty())
```

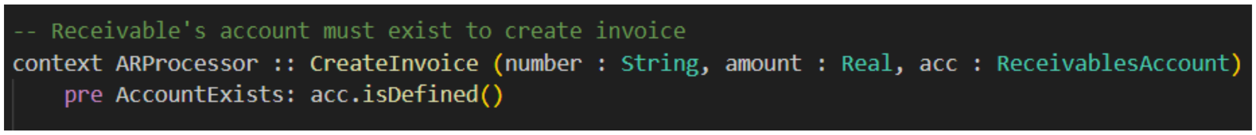
2. Post-condition:



```
-- For all of the arAccounts the following holds:
    -- The collections values is its previous values plus the previous day120Balance and
    -- the day120Balance is the previous day90Balance and
    -- the day90Balance is the previous day60Balance and
    -- the day60Balance is the previous day30Balance and
    -- the day30Balance is the previous currentBalance and
    -- the currentBalance is 0.

-- @pre modifies an identifier to refer to the value it had
-- before the operation executed
context ARProcessor :: EndOfMonthUpdate (arAccounts : Set(ReceivablesAccount))
    post MonthUpdate: arAccounts->forAll(
        currentBalance = 0 and
        day30Balance = currentBalance@pre and
        day60Balance = day30Balance@pre and
        day90Balance = day60Balance@pre and
        day120Balance = day90Balance@pre and
        collections = collections@pre + day120Balance@pre
    )
```

For ARProcessor’s CreateInvoice() operation:

1. Pre-condition:



```
-- Receivable's account must exist to create invoice
context ARProcessor :: CreateInvoice (number : String, amount : Real, acc : ReceivablesAccount)
    pre AccountExists: acc.isDefined()
```

For ReceivablesAccount NewInvoice() operation:

1. Post-condition

```
-----------------------------
context ReceivablesAccount :: NewInvoice ( invoice : Invoice)
    post InvoiceAdded: self.unprocessedInvoices->includes(invoice) -- invoice has been added
```

3.3 Diagrams

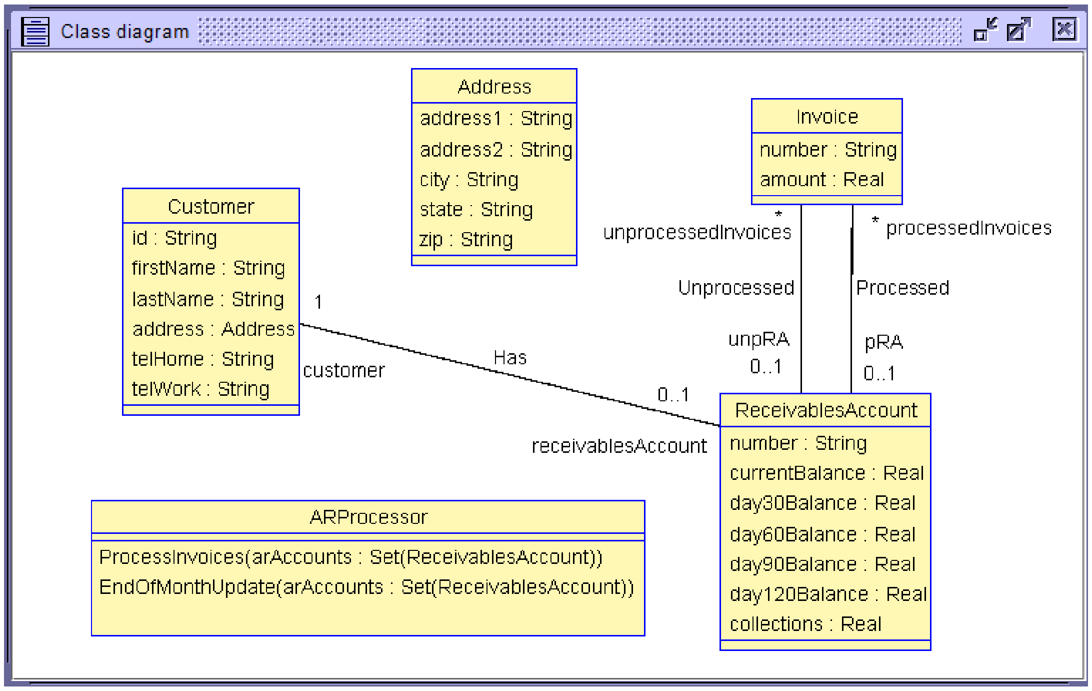


Figure 2. Initial Class Diagram

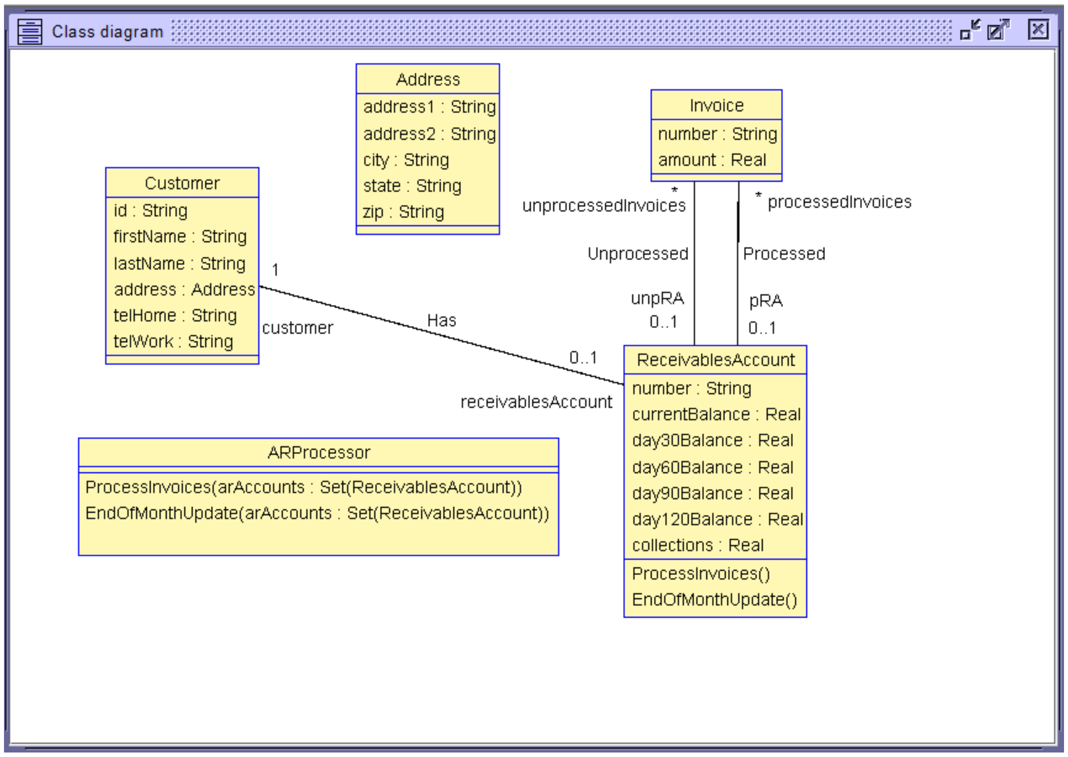


Figure 3. 2nd Version Class Diagram

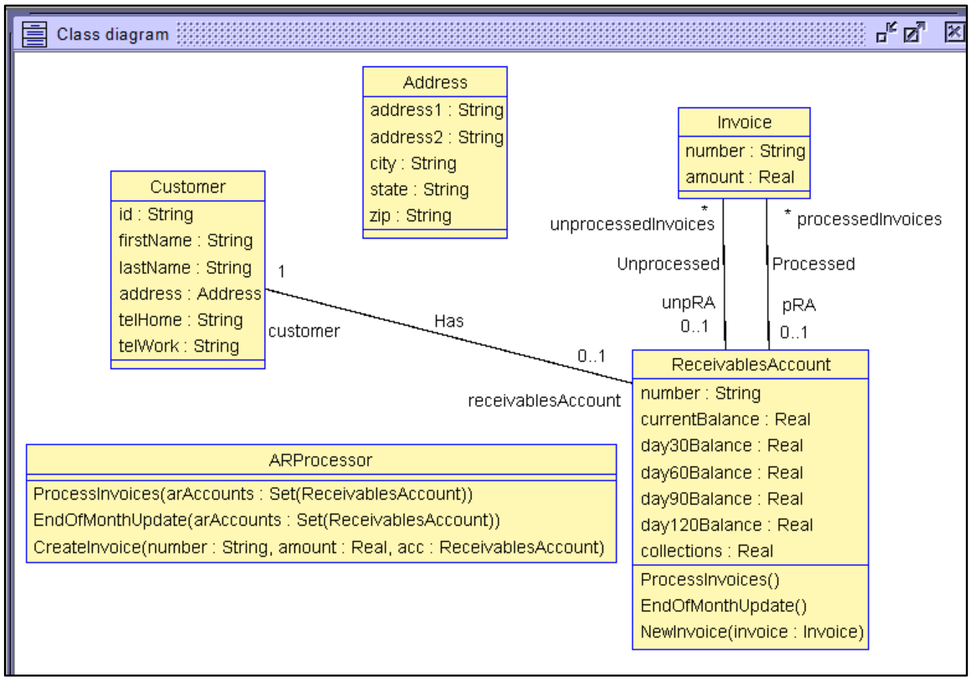


Figure 4. 3rd Version Class Diagram

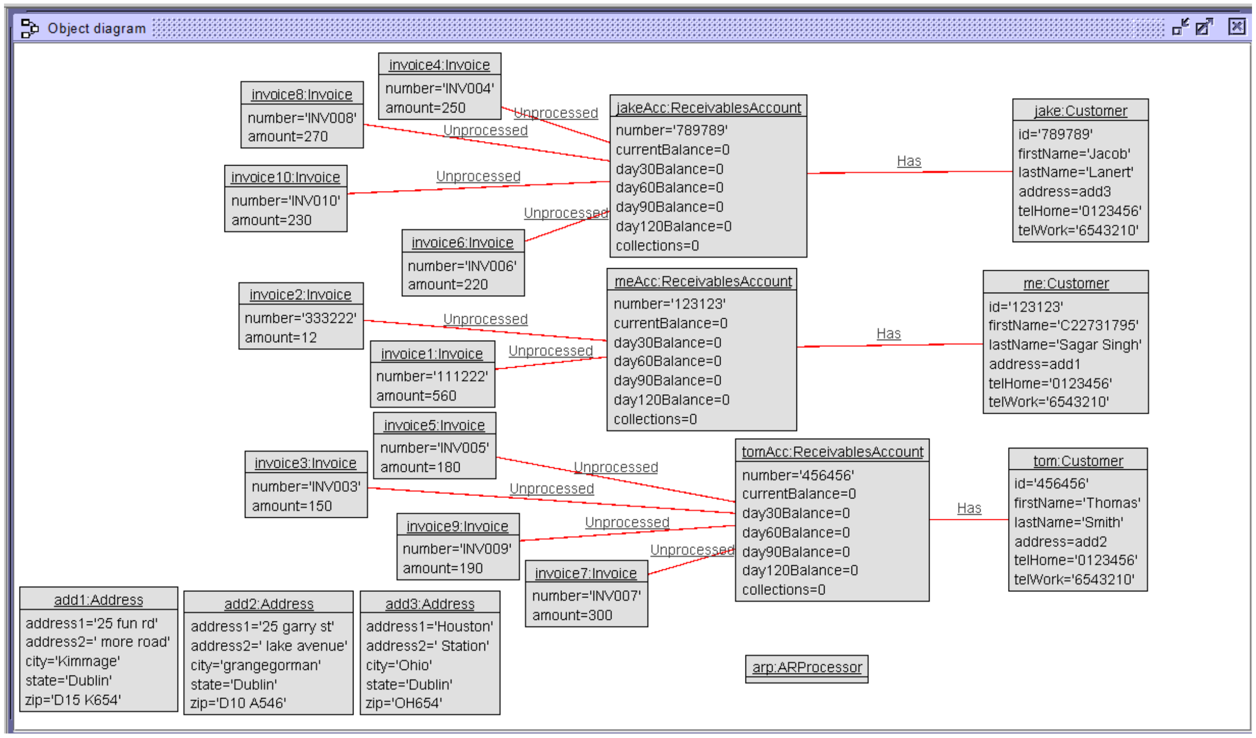


Figure 5. Object Diagram Before Any Operations

3.3.1 Testing main tasks and their results (Object, Sequence, and State Machine diagrams)

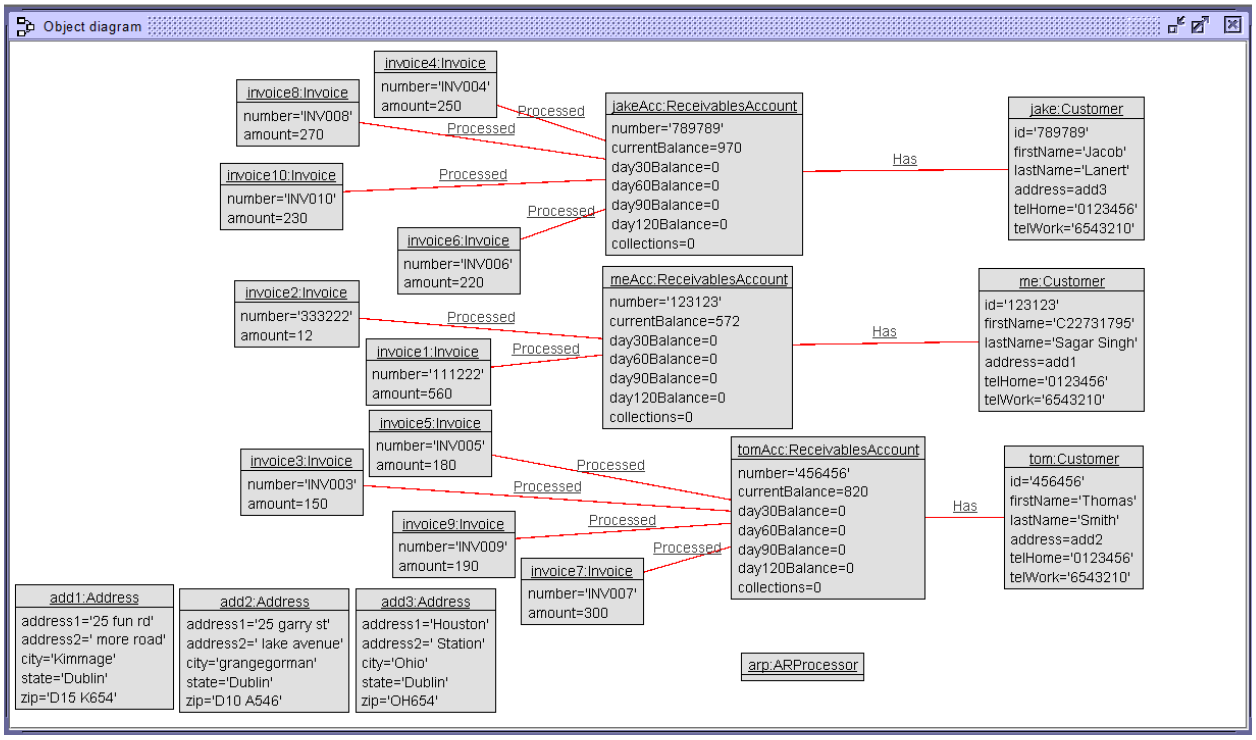


Figure 6. Object Diagram After Processing All Invoices

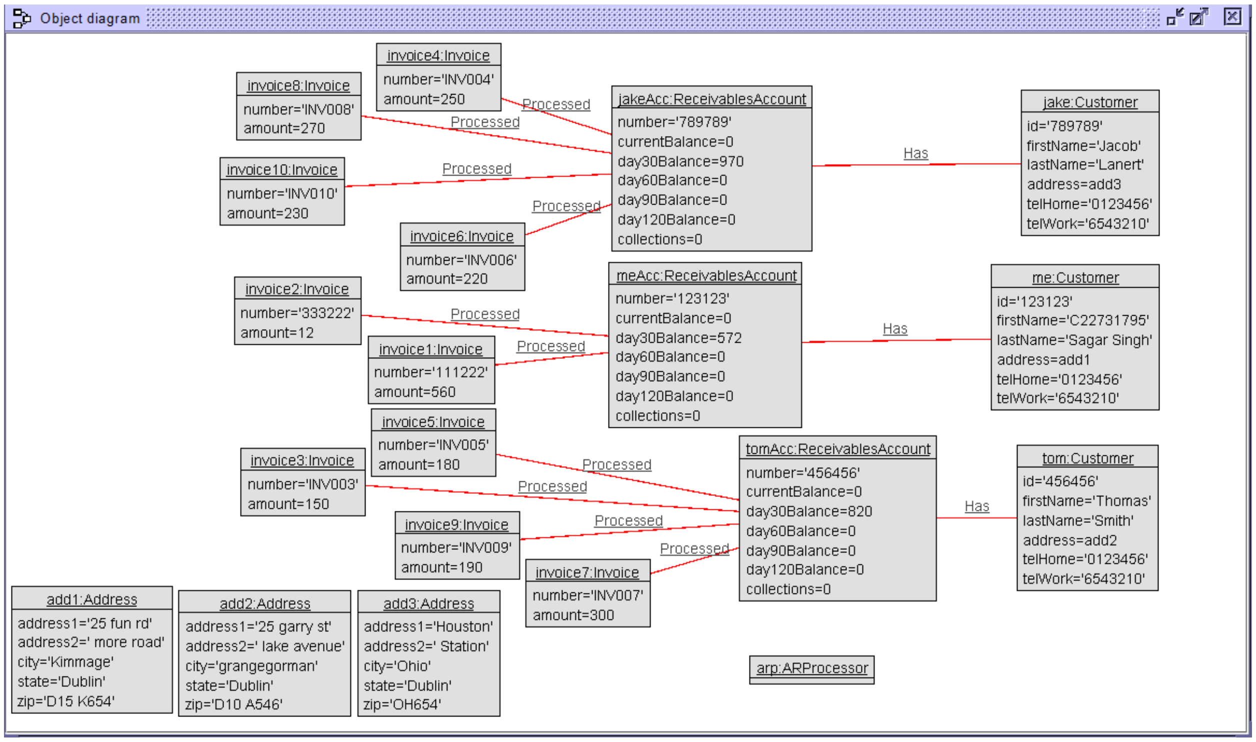


Figure 7. Object Diagram After Processing All Invoices And Updating End Of Month

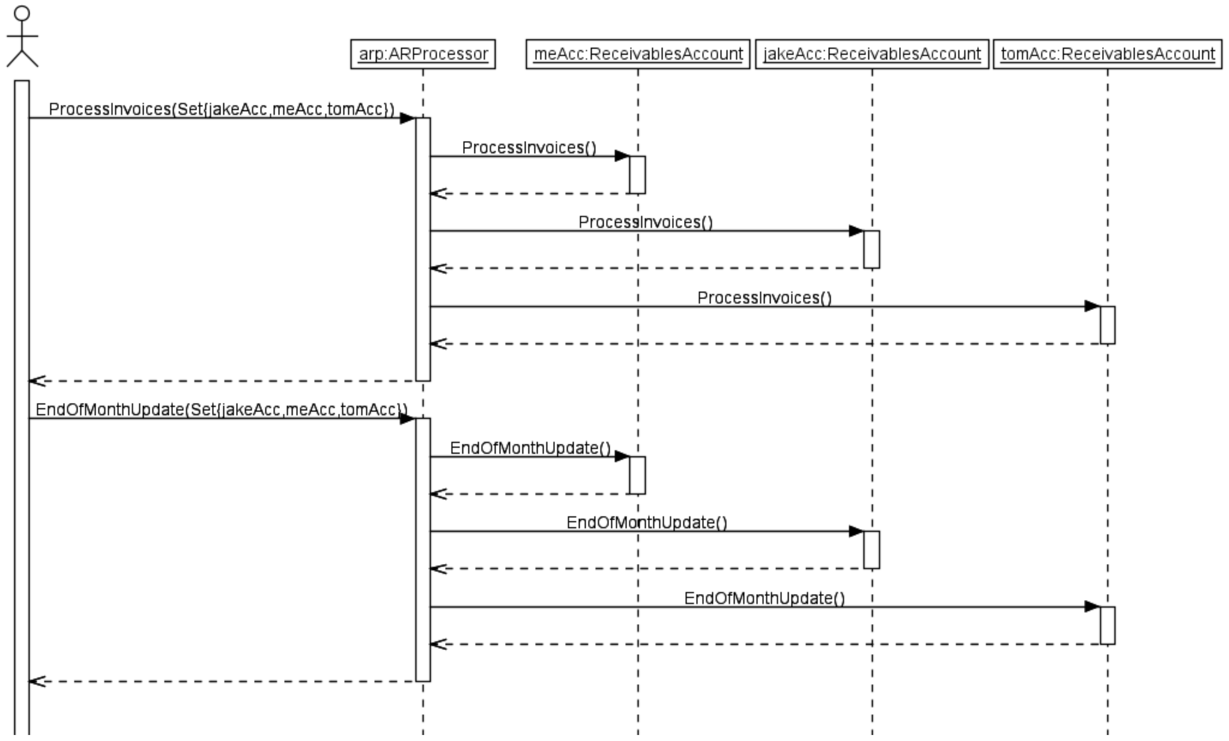


Figure 8. Sequence Diagram of Processing Invoices of All Accounts and Updating End Of Month

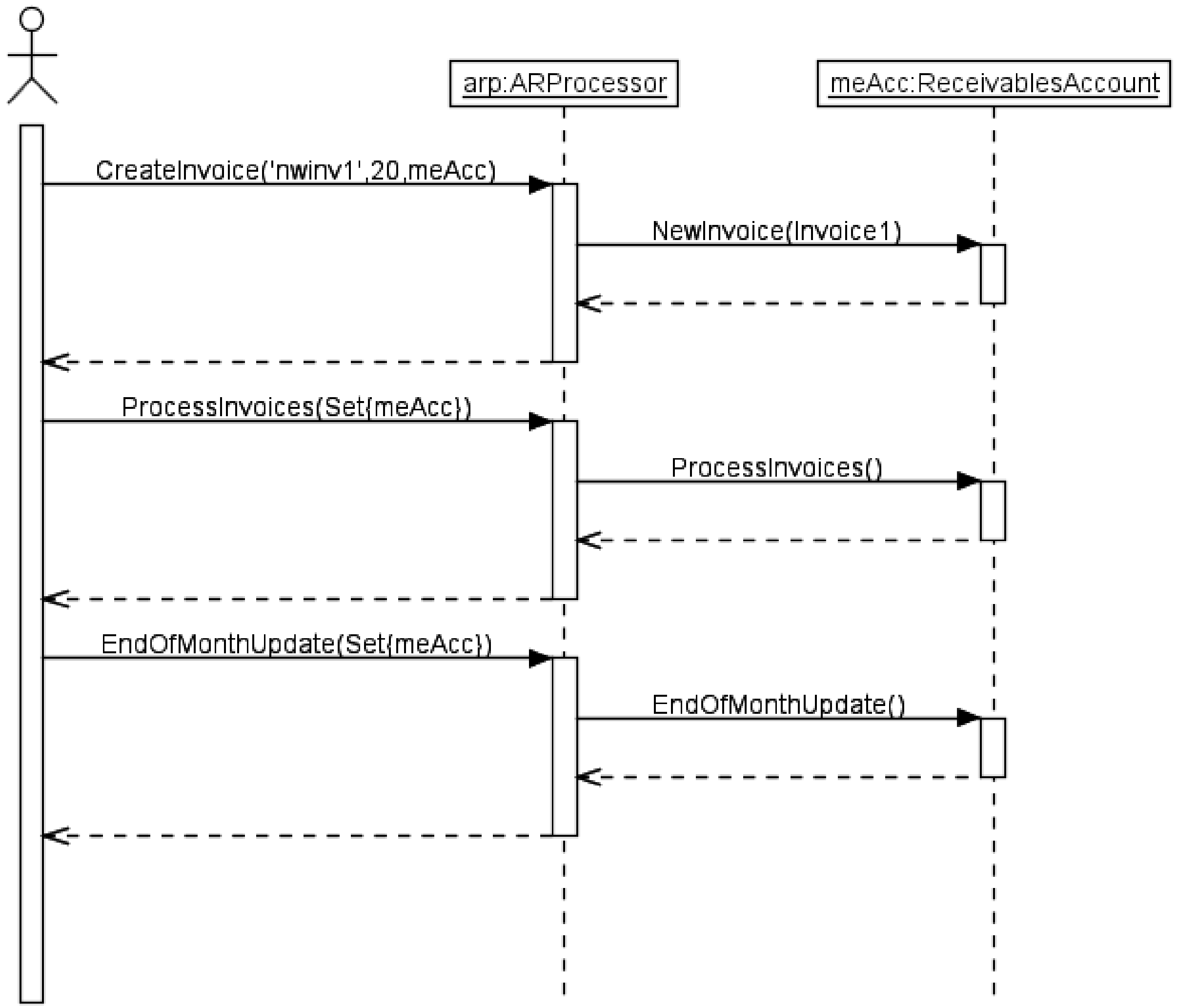


Figure 9. Sequence Diagram of creating invoice for meAcc, then processing all invoices and finally rolling over the balance values with end of month update

Table 1. State Machine of ARProcessor class in various states

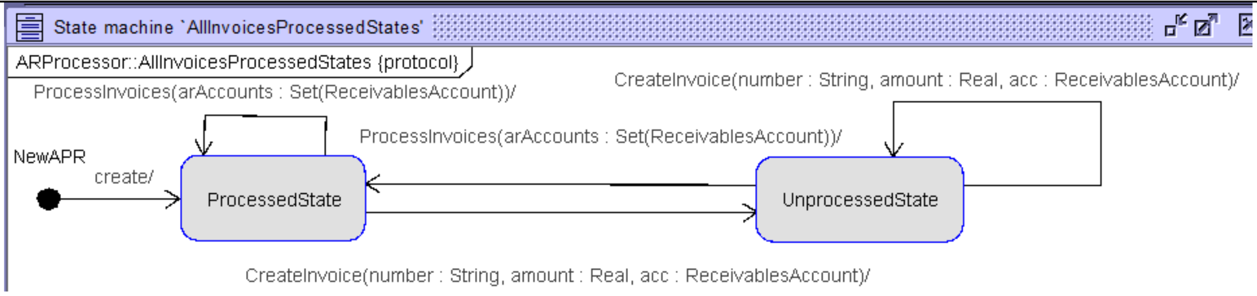
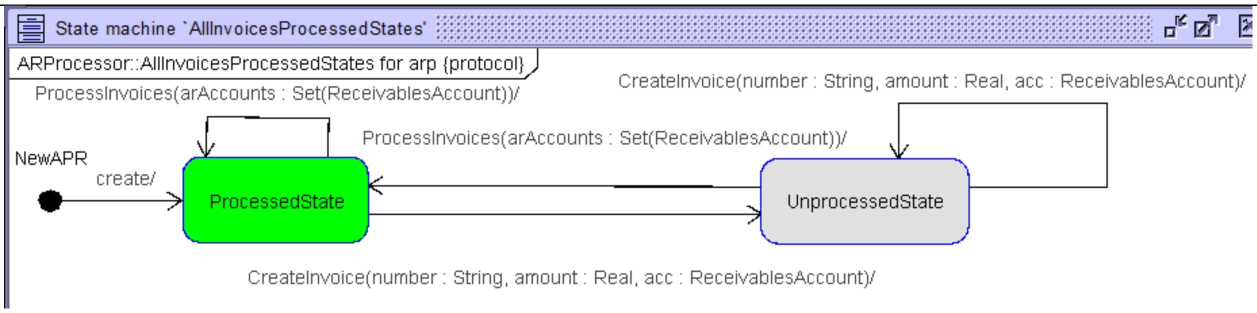
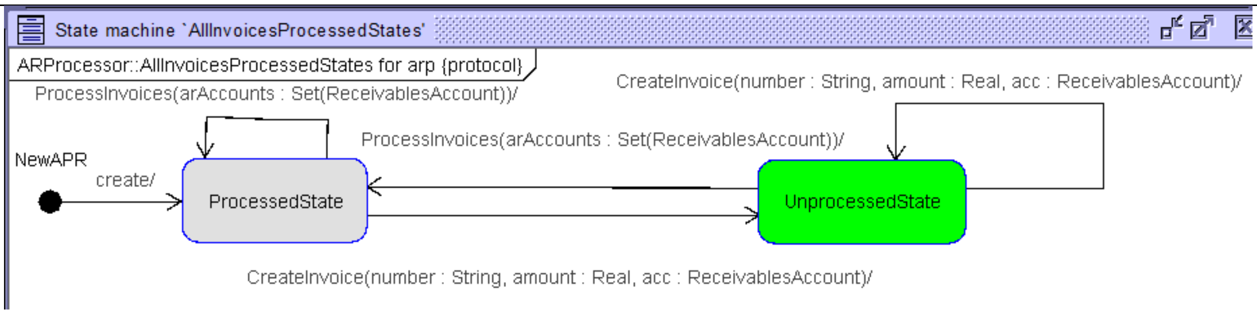
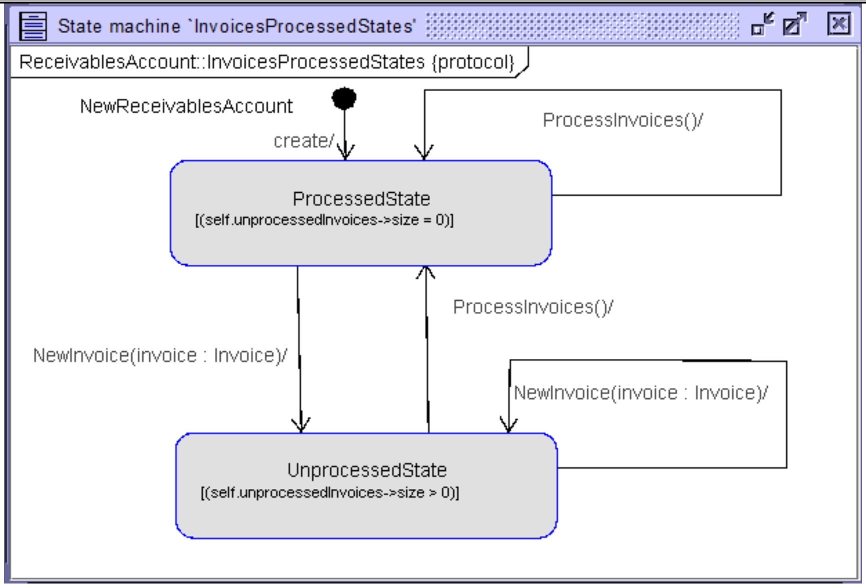
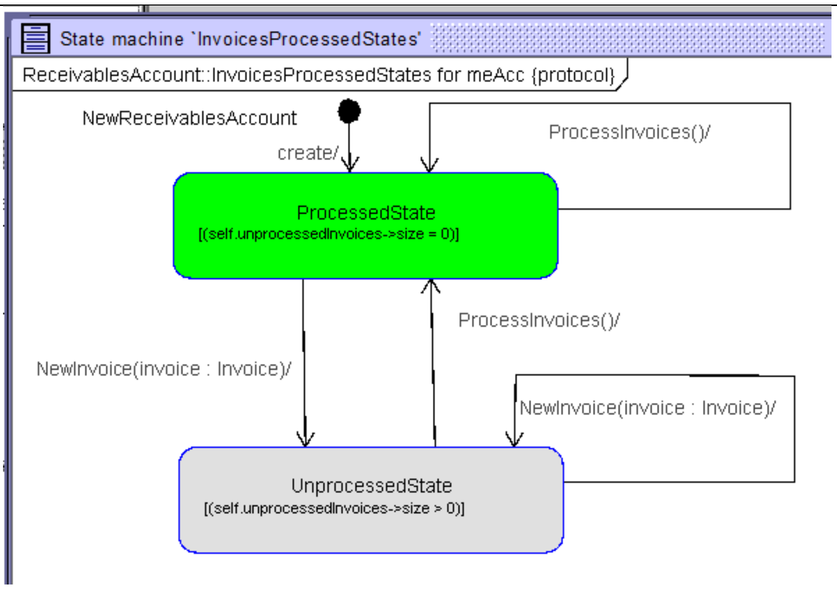
ARProcessor
 State machine `AllInvoicesProcessedStates' ARProcessor::AllInvoicesProcessedStates {protocol} ProcessInvoices(arAccounts : Set(ReceivablesAccount))/ CreateInvoice(number : String, amount : Real, acc : ReceivablesAccount)/ ProcessInvoices(arAccounts : Set(ReceivablesAccount))/ NewAPR create/ ProcessedState UnprocessedState CreateInvoice(number : String, amount : Real, acc : ReceivablesAccount)/ Figure 10. Initial State
 State machine `AllInvoicesProcessedStates' ARProcessor::AllInvoicesProcessedStates for arp {protocol} ProcessInvoices(arAccounts : Set(ReceivablesAccount))/ CreateInvoice(number : String, amount : Real, acc : ReceivablesAccount)/ ProcessInvoices(arAccounts : Set(ReceivablesAccount))/ NewAPR create/ ProcessedState UnprocessedState CreateInvoice(number : String, amount : Real, acc : ReceivablesAccount)/ Figure 11. After Creation or Processing Invoices
 State machine `AllInvoicesProcessedStates' ARProcessor::AllInvoicesProcessedStates for arp {protocol} ProcessInvoices(arAccounts : Set(ReceivablesAccount))/ CreateInvoice(number : String, amount : Real, acc : ReceivablesAccount)/ ProcessInvoices(arAccounts : Set(ReceivablesAccount))/ NewAPR create/ ProcessedState UnprocessedState CreateInvoice(number : String, amount : Real, acc : ReceivablesAccount)/ Figure 12. After Creating an Invoice

Table 2. State Machine ReceivablesAccount class in various states

ReceivablesAccount
 State machine `InvoicesProcessedStates' ReceivablesAccount::InvoicesProcessedStates {protocol} NewReceivablesAccount create/ ProcessInvoices()/ ProcessedState [(self.unprocessedInvoices->size = 0)] ProcessInvoices()/ NewInvoice(invoice : Invoice)/ NewInvoice(invoice : Invoice)/ UnprocessedState [(self.unprocessedInvoices->size > 0)] Figure 13. Initial State
 State machine `InvoicesProcessedStates' ReceivablesAccount::InvoicesProcessedStates for meAcc {protocol} NewReceivablesAccount create/ ProcessInvoices()/ ProcessedState [(self.unprocessedInvoices->size = 0)] ProcessInvoices()/ NewInvoice(invoice : Invoice)/ NewInvoice(invoice : Invoice)/ UnprocessedState [(self.unprocessedInvoices->size > 0)] Figure 14. After Creation or Processing Invoices

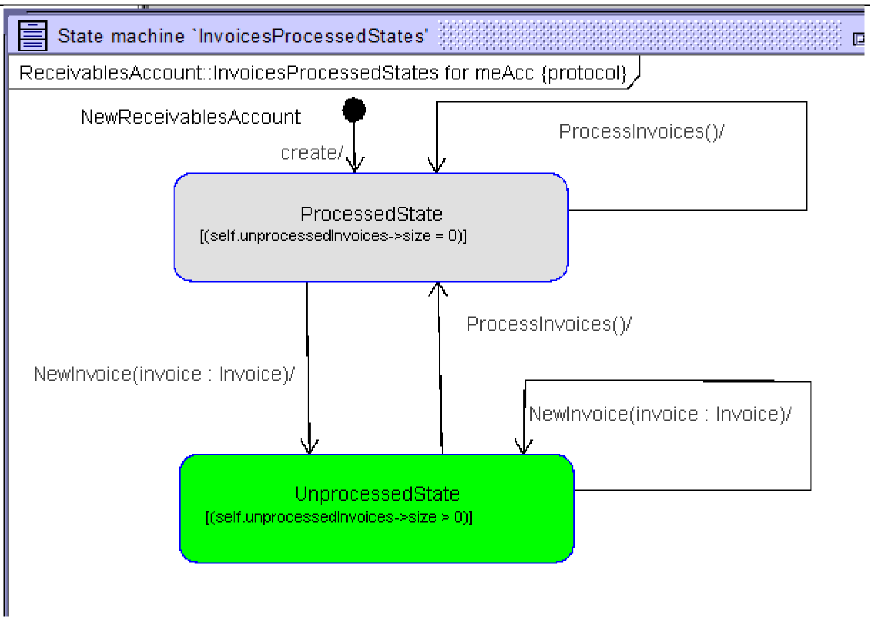

State machine `InvoicesProcessedStates'
ReceivablesAccount::InvoicesProcessedStates for meAcc {protocol}
NewReceivablesAccount
create/
ProcessInvoices()/
ProcessedState
[(self.unprocessedInvoices->size = 0)]
ProcessInvoices()/
NewInvoice(invoice : Invoice)/
NewInvoice(invoice : Invoice)/
UnprocessedState
[(self.unprocessedInvoices->size > 0)]

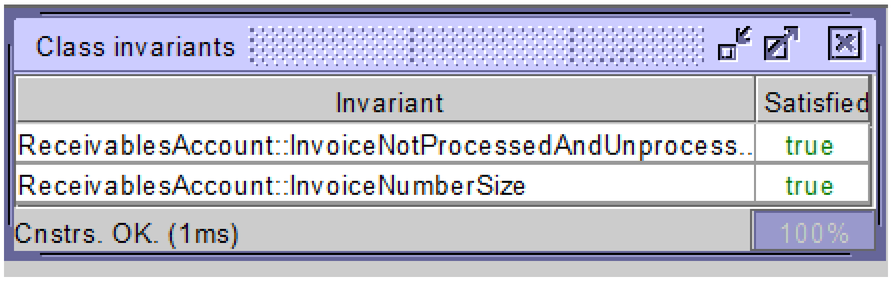
Figure 15. After Creating an Invoice

*Note: Did add a transition from processed state to processed state from processing invoices, since it should do this operation because a set of accounts is passed and only one of them needs to have an unprocessed invoice. Just because a user passes two accounts, one of which does not have any unprocessed invoices, the operation should not cancel. It should only process the account with the unprocessed invoice(s). This is handled in the pre-condition for ARProcessor ProcessInvoices().

4 Testing Constraints

4.1 An invoice being in both processed and unprocessed:

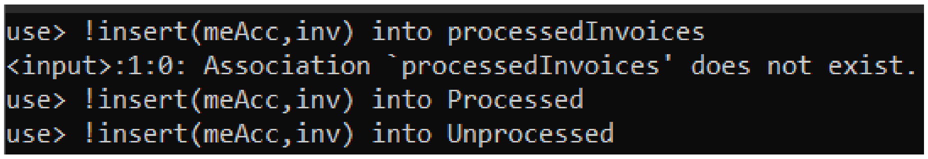
1. Before:



Class invariants

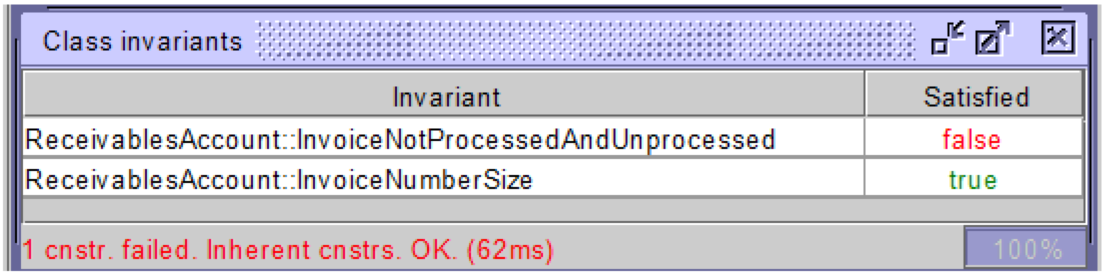
Invariant	Satisfied
ReceivablesAccount::InvoiceNotProcessedAndUnprocess...	true
ReceivablesAccount::InvoiceNumberSize	true

Cnstrs. OK. (1ms) 100%

2. 

```
use> !insert(meAcc,inv) into processedInvoices
<input>:1:0: Association `processedInvoices' does not exist.
use> !insert(meAcc,inv) into Processed
use> !insert(meAcc,inv) into Unprocessed
```

3. After the above SOIL:



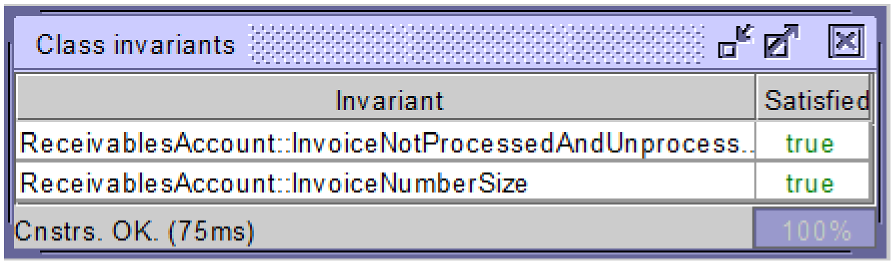
Class invariants

Invariant	Satisfied
ReceivablesAccount::InvoiceNotProcessedAndUnprocessed	false
ReceivablesAccount::InvoiceNumberSize	true

1 cnstr. failed. Inherent cnstrs. OK. (62ms) 100%

4.2 A Receivable's Account not being 6 characters in length:

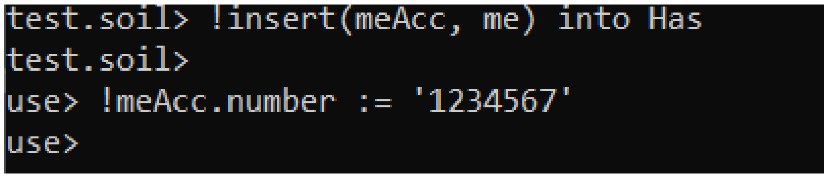
1. Before:



Class invariants

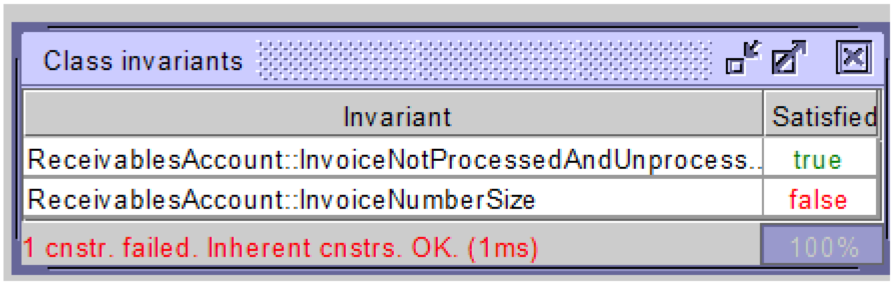
Invariant	Satisfied
ReceivablesAccount::InvoiceNotProcessedAndUnprocess...	true
ReceivablesAccount::InvoiceNumberSize	true

Cnstrs. OK. (75ms) 100%

2. 

```
test.soil> !insert(meAcc, me) into Has
test.soil>
use> !meAcc.number := '1234567'
use>
```

3. After the above SOIL:



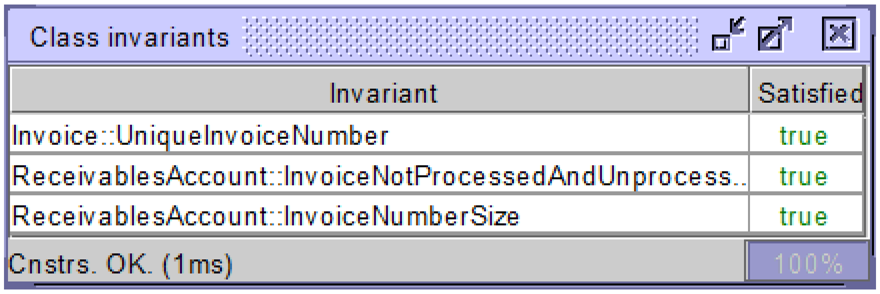
Class invariants

Invariant	Satisfied
ReceivablesAccount::InvoiceNotProcessedAndUnprocess...	true
ReceivablesAccount::InvoiceNumberSize	false

1 cnstr. failed. Inherent cnstrs. OK. (1ms) 100%

4.3 Invoices must have uniquely identifiable numbers

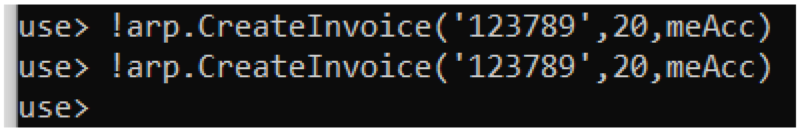
1. Before:



Class invariants

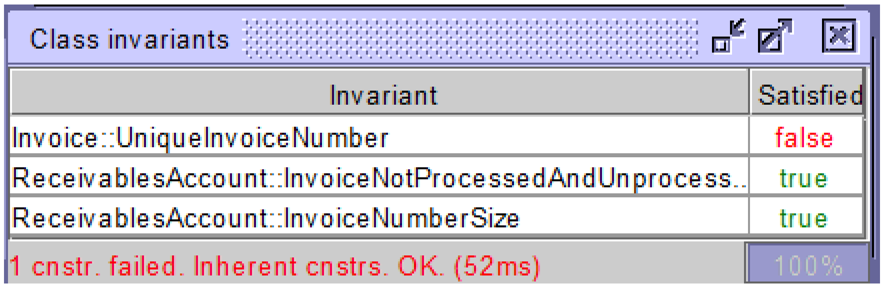
Invariant	Satisfied
Invoice::UniqueInvoiceNumber	true
ReceivablesAccount::InvoiceNotProcessedAndUnprocess..	true
ReceivablesAccount::InvoiceNumberSize	true

Cnstrs. OK. (1ms) 100%



```
use> !arp.CreateInvoice('123789',20,meAcc)
use> !arp.CreateInvoice('123789',20,meAcc)
use>
```

2. After:



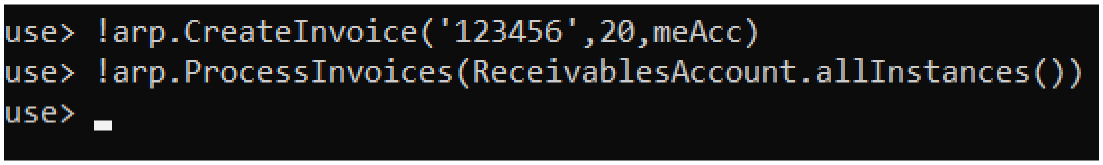
Class invariants

Invariant	Satisfied
Invoice::UniqueInvoiceNumber	false
ReceivablesAccount::InvoiceNotProcessedAndUnprocess..	true
ReceivablesAccount::InvoiceNumberSize	true

1 cnstr. failed. Inherent cnstrs. OK. (52ms) 100%

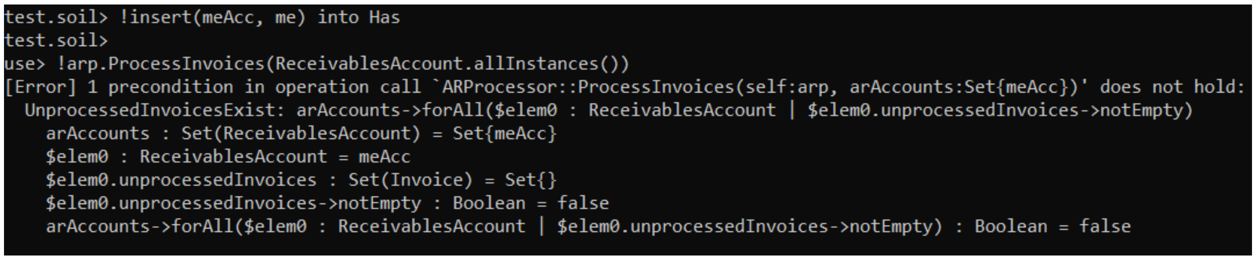
4.4 Testing ARProcessor's ProcessInvoices():

1. Pre-condition that there must be some unprocessed invoices:
   a. Working:



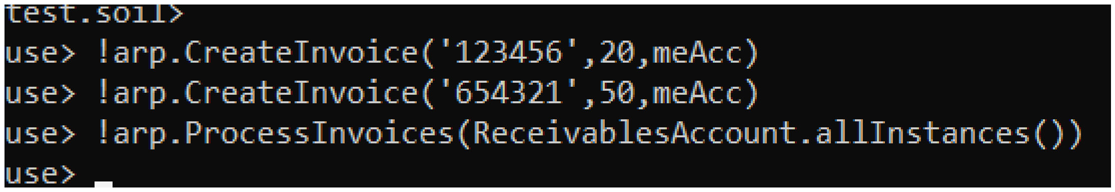
```
use> !arp.CreateInvoice('123456',20,meAcc)
use> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
use>
```

   b. Failed: failed since there is no unprocessed invoices at all



```
test.soil> !insert(meAcc, me) into Has
test.soil>
use> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
[Error] 1 precondition in operation call `ARProcessor::ProcessInvoices(self:arp, arAccounts:Set{meAcc})' does not hold:
  UnprocessedInvoicesExist: arAccounts->forAll($elem0 : ReceivablesAccount | $elem0.unprocessedInvoices->notEmpty)
    arAccounts : Set(ReceivablesAccount) = Set{meAcc}
    $elem0 : ReceivablesAccount = meAcc
    $elem0.unprocessedInvoices : Set(Invoice) = Set{}
    $elem0.unprocessedInvoices->notEmpty : Boolean = false
    arAccounts->forAll($elem0 : ReceivablesAccount | $elem0.unprocessedInvoices->notEmpty) : Boolean = false
```

2. Post-condition that set unprocessed invoices become processed invoices
    a. Working:

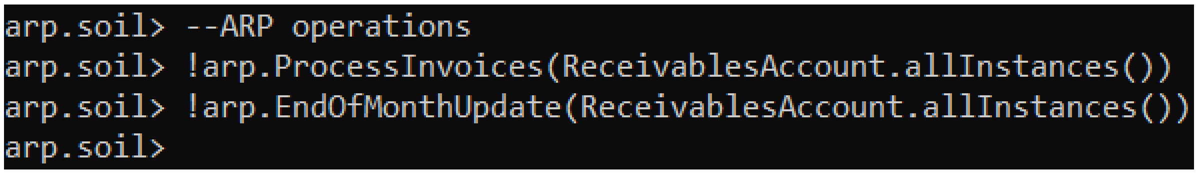


```
test.soil>
use> !arp.CreateInvoice('123456',20,meAcc)
use> !arp.CreateInvoice('654321',50,meAcc)
use> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
use>
```

    b. Failed: Cannot fail this on purpose, since the operation deletes association for all accounts passed in unprocessed and inserts them into processed set.

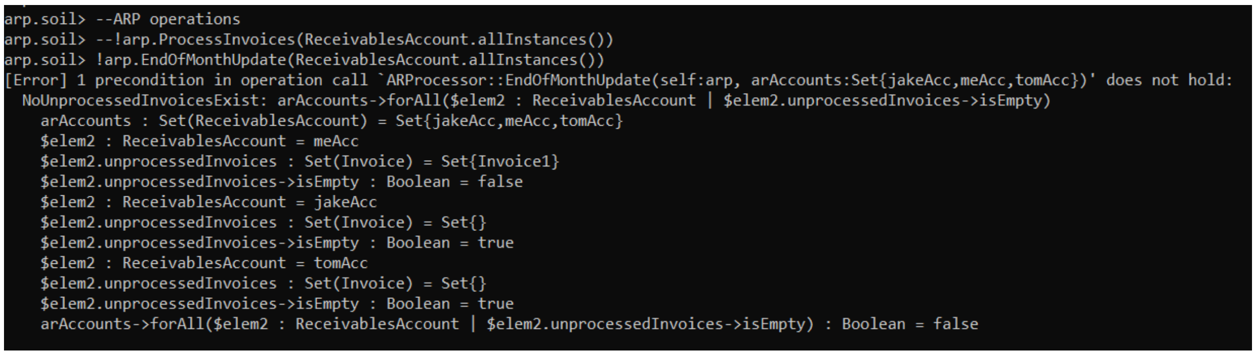
4.5 Testing ARProcessor's EndOfMonthUpdate():

1. Pre-condition that there are no unprocessed invoices:
    a. Working:



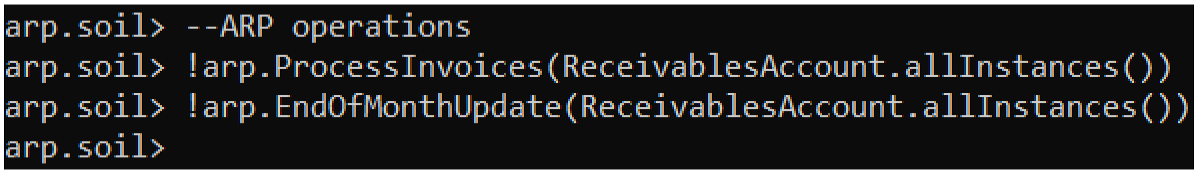
```
arp.soil> --ARP operations
arp.soil> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
arp.soil> !arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
arp.soil>
```

    b. Failed: There is an unprocessed invoice still. Cannot call this before all passed accounts have been processed.



```
arp.soil> --ARP operations
arp.soil> --!arp.ProcessInvoices(ReceivablesAccount.allInstances())
arp.soil> !arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
[Error] 1 precondition in operation call `ARProcessor::EndOfMonthUpdate(self:arp, arAccounts:Set{jakeAcc,meAcc,tomAcc})' does not hold:
  NoUnprocessedInvoicesExist: arAccounts->forAll($elem2 : ReceivablesAccount | $elem2.unprocessedInvoices->isEmpty)
    arAccounts : Set(ReceivablesAccount) = Set{jakeAcc,meAcc,tomAcc}
    $elem2 : ReceivablesAccount = meAcc
    $elem2.unprocessedInvoices : Set(Invoice) = Set{Invoice1}
    $elem2.unprocessedInvoices->isEmpty : Boolean = false
    $elem2 : ReceivablesAccount = jakeAcc
    $elem2.unprocessedInvoices : Set(Invoice) = Set{}
    $elem2.unprocessedInvoices->isEmpty : Boolean = true
    $elem2 : ReceivablesAccount = tomAcc
    $elem2.unprocessedInvoices : Set(Invoice) = Set{}
    $elem2.unprocessedInvoices->isEmpty : Boolean = true
    arAccounts->forAll($elem2 : ReceivablesAccount | $elem2.unprocessedInvoices->isEmpty) : Boolean = false
```

2. Post-condition that for all balances roll over to next by
    a. Current balance -> 30 day -> 60 day -> 90 day ->120 day -> collections
    b. current balance reset to zero
    c. collections adds previous collections sum and 120 day balance
    d. Working:

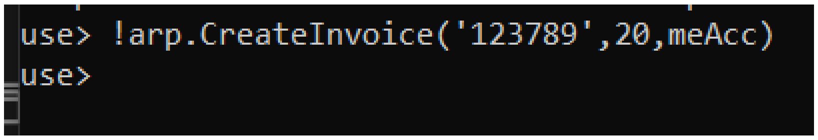


```
arp.soil> --ARP operations
arp.soil> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
arp.soil> !arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
arp.soil>
```

    e. Failed: Cannot fail since it just checking that the functionality of the operation is correct, which it is always is inherently.

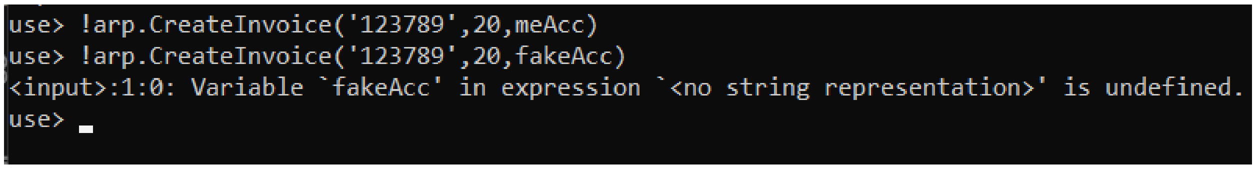
4.6 Testing ARProcessor's CreateInvoice():

1. Pre-condition: that the new invoice being created must have an existing account to associate with, otherwise the invoice is considered not viable
    a. Working:



```
use> !arp.CreateInvoice('123789',20,meAcc)
use>
```

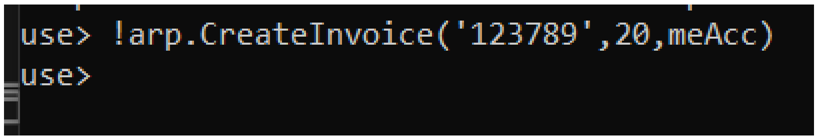
    b. Failed: here the command shell already catches that the object is undefined



```
use> !arp.CreateInvoice('123789',20,meAcc)
use> !arp.CreateInvoice('123789',20,fakeAcc)
<input>:1:0: Variable `fakeAcc' in expression `<no string representation>' is undefined.
use>
```

4.7 Testing ReceivablesAccount NewInvoice():

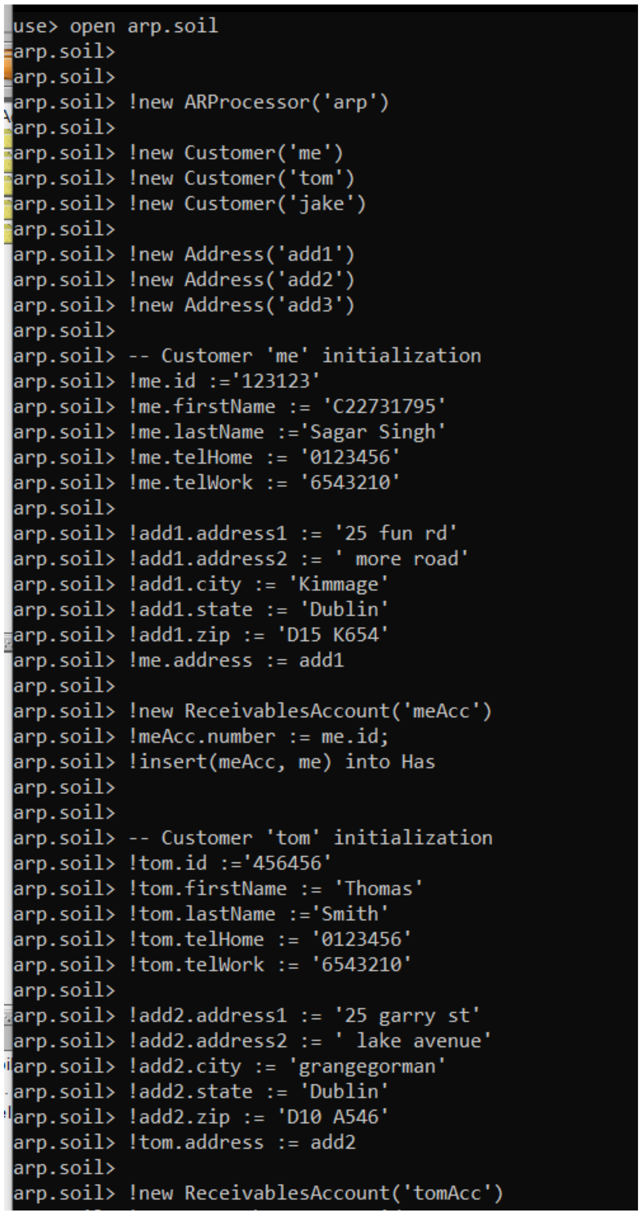
1. Post-condition: that invoice has been added to new unprocessed set
    a. Working: Since CreateInvoice() calls NewInvoice() in ReceivablesAccount



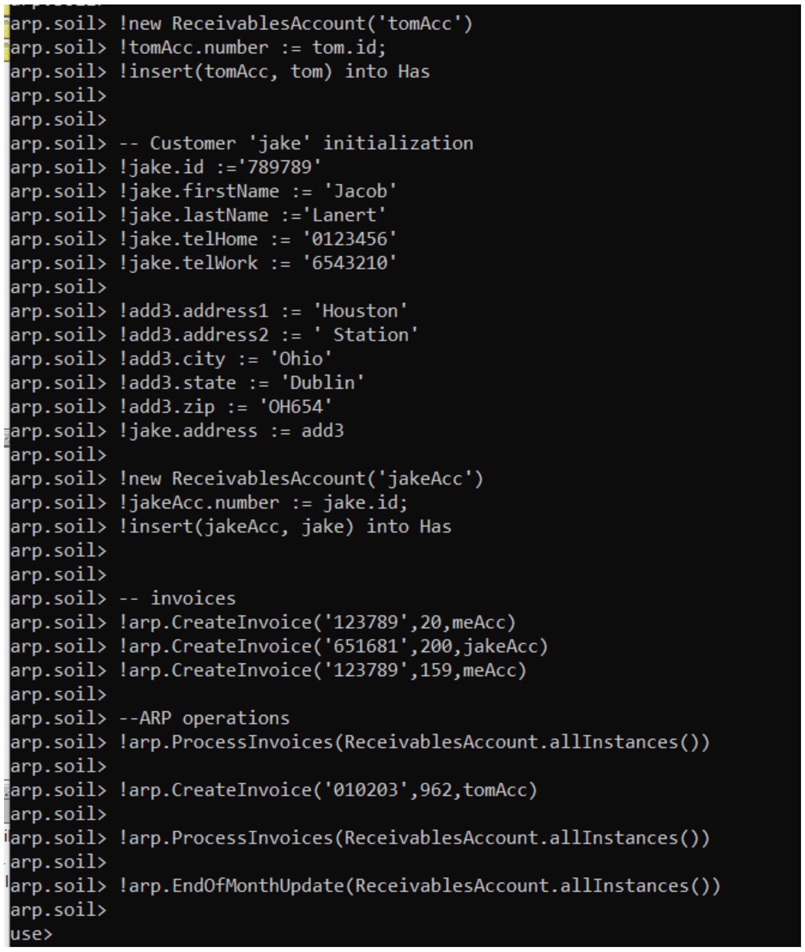
```
use> !arp.CreateInvoice('123789',20,meAcc)
use>
```

    b. Failed: Cannot inherently fail since the functionality of the operation does add it to the association set.

5 Testing With arp.soil File

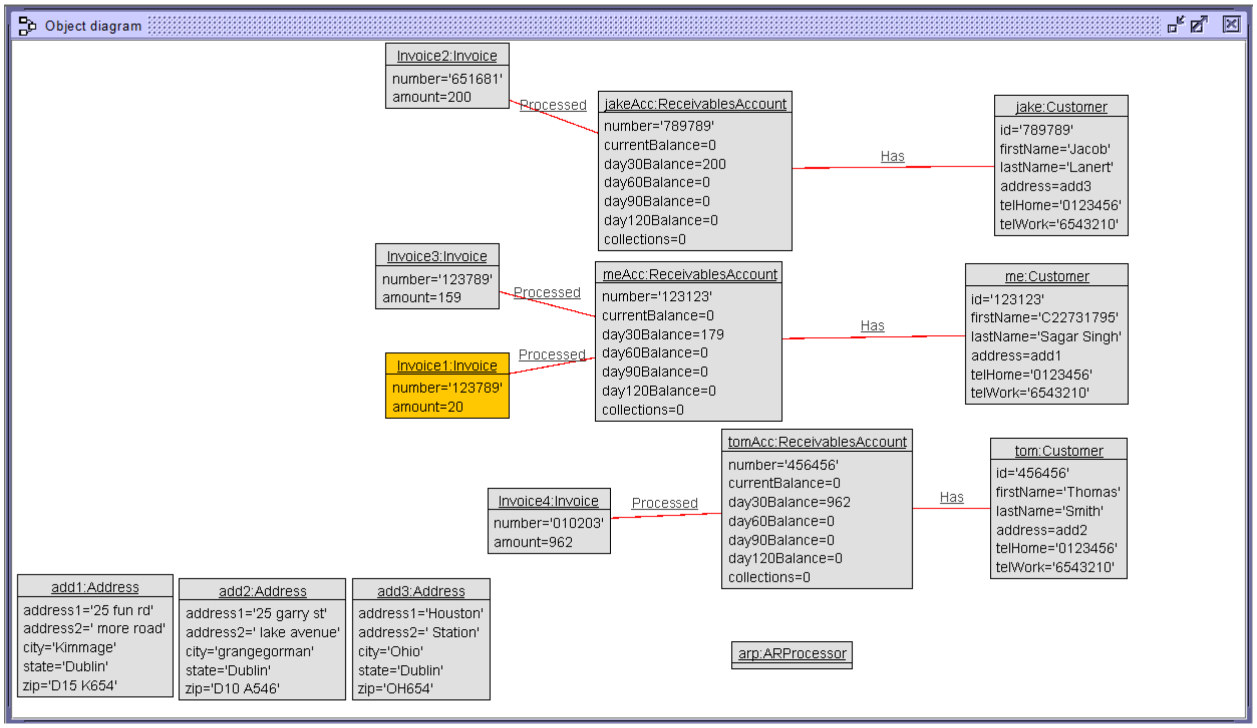


```
use> open arp.soil
arp.soil>
arp.soil>
arp.soil> !new ARProcessor('arp')
arp.soil>
arp.soil> !new Customer('me')
arp.soil> !new Customer('tom')
arp.soil> !new Customer('jake')
arp.soil>
arp.soil> !new Address('add1')
arp.soil> !new Address('add2')
arp.soil> !new Address('add3')
arp.soil>
arp.soil> -- Customer 'me' initialization
arp.soil> !me.id :='123123'
arp.soil> !me.firstName := 'C22731795'
arp.soil> !me.lastName :='Sagar Singh'
arp.soil> !me.telHome := '0123456'
arp.soil> !me.telWork := '6543210'
arp.soil>
arp.soil> !add1.address1 := '25 fun rd'
arp.soil> !add1.address2 := ' more road'
arp.soil> !add1.city := 'Kimmage'
arp.soil> !add1.state := 'Dublin'
arp.soil> !add1.zip := 'D15 K654'
arp.soil> !me.address := add1
arp.soil>
arp.soil> !new ReceivablesAccount('meAcc')
arp.soil> !meAcc.number := me.id;
arp.soil> !insert(meAcc, me) into Has
arp.soil>
arp.soil>
arp.soil> -- Customer 'tom' initialization
arp.soil> !tom.id :='456456'
arp.soil> !tom.firstName := 'Thomas'
arp.soil> !tom.lastName :='Smith'
arp.soil> !tom.telHome := '0123456'
arp.soil> !tom.telWork := '6543210'
arp.soil>
arp.soil> !add2.address1 := '25 garry st'
arp.soil> !add2.address2 := ' lake avenue'
arp.soil> !add2.city := 'grangegorman'
arp.soil> !add2.state := 'Dublin'
arp.soil> !add2.zip := 'D10 A546'
arp.soil> !tom.address := add2
arp.soil>
arp.soil> !new ReceivablesAccount('tomAcc')
```



```
arp.soil> !new ReceivablesAccount('tomAcc')
arp.soil> !tomAcc.number := tom.id;
arp.soil> !insert(tomAcc, tom) into Has
arp.soil>
arp.soil>
arp.soil> -- Customer 'jake' initialization
arp.soil> !jake.id :='789789'
arp.soil> !jake.firstName := 'Jacob'
arp.soil> !jake.lastName :='Lanert'
arp.soil> !jake.telHome := '0123456'
arp.soil> !jake.telWork := '6543210'
arp.soil>
arp.soil> !add3.address1 := 'Houston'
arp.soil> !add3.address2 := ' Station'
arp.soil> !add3.city := 'Ohio'
arp.soil> !add3.state := 'Dublin'
arp.soil> !add3.zip := 'OH654'
arp.soil> !jake.address := add3
arp.soil>
arp.soil> !new ReceivablesAccount('jakeAcc')
arp.soil> !jakeAcc.number := jake.id;
arp.soil> !insert(jakeAcc, jake) into Has
arp.soil>
arp.soil>
arp.soil> -- invoices
arp.soil> !arp.CreateInvoice('123789',20,meAcc)
arp.soil> !arp.CreateInvoice('651681',200,jakeAcc)
arp.soil> !arp.CreateInvoice('123789',159,meAcc)
arp.soil>
arp.soil> --ARP operations
arp.soil> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
arp.soil>
arp.soil> !arp.CreateInvoice('010203',962,tomAcc)
arp.soil>
arp.soil> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
arp.soil>
arp.soil> !arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
arp.soil>
use>
```

Resulting Object Diagram:



Invoice2:Invoice
number='651681'
amount=200

jakeAcc:ReceivablesAccount
number='789789'
currentBalance=0
day30Balance=200
day60Balance=0
day90Balance=0
day120Balance=0
collections=0

jake:Customer
id='789789'
firstName='Jacob'
lastName='Lanert'
address=add3
telHome='0123456'
telWork='6543210'

Invoice3:Invoice
number='123789'
amount=159

meAcc:ReceivablesAccount
number='123123'
currentBalance=0
day30Balance=179
day60Balance=0
day90Balance=0
day120Balance=0
collections=0

me:Customer
id='123123'
firstName='C22731795'
lastName='Sagar Singh'
address=add1
telHome='0123456'
telWork='6543210'

Invoice1:Invoice
number='123789'
amount=20

tomAcc:ReceivablesAccount
number='456456'
currentBalance=0
day30Balance=962
day60Balance=0
day90Balance=0
day120Balance=0
collections=0

tom:Customer
id='456456'
firstName='Thomas'
lastName='Smith'
address=add2
telHome='0123456'
telWork='6543210'

Invoice4:Invoice
number='010203'
amount=962

add1:Address
address1='25 fun rd'
address2=' more road'
city='Kimmage'
state='Dublin'
zip='D15 K654'

add2:Address
address1='25 garry st'
address2=' lake avenue'
city='grangegorman'
state='Dublin'
zip='D10 A546'

add3:Address
address1='Houston'
address2=' Station'
city='Ohio'
state='Dublin'
zip='OH654'

arp:ARProcessor

Figure 16. Object diagram after executing arp.soil file

6 Evaluation and Conclusion

All the set-out goals and plans for model have been successfully implemented and extensively tested with different cases showing how the operations with the constraints are working as intended to catch the wrong operations and successfully pass the correctly executed operations.

Demonstrated extensively the workings of the model through sequence, changing object and state machines diagrams. In addition, showcased that successful SOIL operations and intentional faulty operations were caught and shown through command prompt window snippets.

7 References

Frankel, D. S. (2003). *Model Drive Architecture - Applying MDA to Enterprise Computing*. Nashville, TN: John Wiley & Sons.

8 Appendix

8.1 .use code

```
model AccountsReceivableProcessor

--------------
--Classes
--------------


--Main Class
class ReceivablesAccount
    attributes
        number : String
        currentBalance : Real init = 0
        day30Balance : Real init = 0
        day60Balance : Real init = 0
        day90Balance : Real init = 0
        day120Balance : Real init = 0
        collections : Real init = 0
    operations
        ProcessInvoices()
        begin
            for invoice in self.unprocessedInvoices do
                    self.currentBalance := self.currentBalance + invoice.amount;
                    insert(self, invoice) into Processed;
                    delete(self, invoice) from Unprocessed;
            end
        end

        EndOfMonthUpdate()
        begin
                self.collections := self.collections + self.day120Balance;
                self.day120Balance := self.day90Balance;
                self.day90Balance := self.day60Balance;
                self.day60Balance := self.day30Balance;
                self.day30Balance := self.currentBalance;
                self.currentBalance := 0;
        end

        NewInvoice( invoice : Invoice )
        begin
            insert(self, invoice) into Unprocessed;
        end

    statemachines
        psm InvoicesProcessedStates
        states
            NewReceivablesAccount : initial
            UnprocessedState [self.unprocessedInvoices->size() > 0]
            ProcessedState [self.unprocessedInvoices->size() = 0]
        transitions
            NewReceivablesAccount -> ProcessedState { create }
            UnprocessedState -> ProcessedState { ProcessInvoices() }
            ProcessedState -> UnprocessedState { NewInvoice() }
            UnprocessedState -> UnprocessedState { NewInvoice() }
            ProcessedState -> ProcessedState { ProcessInvoices() }
        end
end

class ARProcessor
    operations
        ProcessInvoices(arAccounts : Set(ReceivablesAccount))
        begin
            for recAcc in arAccounts do
                recAcc.ProcessInvoices();
            end
        end

        EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
        begin
            for recAcc in arAccounts do
                recAcc.EndOfMonthUpdate();
            end
        end

        CreateInvoice(number : String, amount : Real, acc : ReceivablesAccount)
        begin
            declare newInvoice : Invoice;
            newInvoice := new Invoice;
            newInvoice.number := number;
            newInvoice.amount := amount;
            acc.NewInvoice(newInvoice);
        end

        test()

    statemachines
        psm AllInvoicesProcessedStates
        states
            NewAPR : initial
            UnprocessedState
            ProcessedState
        transitions
            NewAPR -> ProcessedState { create }
            UnprocessedState -> ProcessedState { ProcessInvoices() }
            ProcessedState -> UnprocessedState { CreateInvoice() }
            UnprocessedState -> UnprocessedState { CreateInvoice() }
            ProcessedState -> ProcessedState { ProcessInvoices() }
        end
end

class Customer
    attributes
    id : String
    firstName : String
    lastName : String
    address : Address
    telHome : String
    telWork : String
end

class Address
    attributes
        address1 : String
        address2 : String
        city : String
        state : String
        zip : String
end

class Invoice
    attributes
        number : String
        amount : Real
end


-----------
-- Associations
-----------
association Has between
    ReceivablesAccount[0..1]
    Customer[1] role customer
end

association Unprocessed between
    ReceivablesAccount[0..1] role unpRA
    Invoice[0..*] role unprocessedInvoices
end

association Processed between
    ReceivablesAccount[0..1] role pRA
    Invoice[0..*] role processedInvoices
end


-----------
-- Constraints
-----------
constraints

------------------------------
-- ReceivablesAccount invariants
------------------------------
-- An Invoice cannot be both unprocessed and processed.
context ReceivablesAccount
    inv InvoiceNotProcessedAndUnprocessed: unprocessedInvoices-
>intersection(processedInvoices)->isEmpty()

-- An invoice number must be six characters in length.
context ReceivablesAccount
    inv InvoiceNumberSize: self.number.size() = 6

--- Invoice number must be unique
context Invoice
    inv UniqueInvoiceNumber: Invoice.allInstances()->forAll(e, j | e <> j implies
e.number <> j.number)
```

```
------------------------------
-- ARProcessor :: ProcessInvoices pre-conditions
------------------------------
-- There must be some unprocessedInvoices from the set passed.
context ARProcessor :: ProcessInvoices(arAccounts : Set(ReceivablesAccount))
    pre UnprocessedInvoicesExist: arAccounts->exists(unprocessedInvoices-
>notEmpty())


------------------------------
-- ARProcessor :: ProcessInvoices post-conditions
------------------------------
-- unprocessedInvoices become processedInvoices
context ARProcessor :: ProcessInvoices( arAccounts : Set(ReceivablesAccount))
    post AllInvoicesProcessed: arAccounts->forAll(
        unprocessedInvoices->isEmpty() and
        processedInvoices = processedInvoices@pre->union(unprocessedInvoices@pre)
    )


------------------------------
-- ARProcessor :: EndOfMonthUpdate pre-conditions
------------------------------
-- There are no unprocessed invoices.
context ARProcessor :: EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
    pre NoUnprocessedInvoicesExist: arAccounts->forAll(unprocessedInvoices-
>isEmpty())


------------------------------
-- ARProcessor :: EndOfMonthUpdate post-conditions
------------------------------
-- For all of the arAccounts the following holds:
    -- The collections values is its previous values plus the previous
day120Balance and
    -- the day120Balance is the previous day90Balance and
    -- the day90Balance is the previous day60Balance and
    -- the day60Balance is the previous day30Balance and
    -- the day30Balance is the previous currentBalance and
    -- the currentBalance is 0.

-- @pre modifies an identifier to refer to the value it had
-- before the operation executed
```

```
context ARProcessor :: EndOfMonthUpdate (arAccounts : Set(ReceivablesAccount))
    post MonthUpdate: arAccounts->forAll(
        currentBalance = 0 and
        day30Balance = currentBalance@pre and
        day60Balance = day30Balance@pre and
        day90Balance = day60Balance@pre and
        day120Balance = day90Balance@pre and
        collections = collections@pre + day120Balance@pre
    )


------------------------------
-- ARProcessor :: CreateInvoice pre-conditions
------------------------------
-- Receivable's account must exist to create invoice
context ARProcessor :: CreateInvoice (number : String, amount : Real, acc :
ReceivablesAccount)
    pre AccountExists: acc.isDefined()

------------------------------
-- ReceivablesAccount :: NewInvoice post-conditions
------------------------------
context ReceivablesAccount :: NewInvoice ( invoice : Invoice)
    post InvoiceAdded: self.unprocessedInvoices->includes(invoice) -- invoice has
been added
```

8.2 .soil

```


!new ARProcessor('arp')

!new Customer('me')
!new Customer('tom')
!new Customer('jake')

!new Address('add1')
!new Address('add2')
!new Address('add3')

-- Customer 'me' initialization
!me.id :='123123'
!me.firstName := 'C22731795'
!me.lastName :='Sagar Singh'
```

```
!me.telHome := '0123456'
!me.telWork := '6543210'

!add1.address1 := '25 fun rd'
!add1.address2 := ' more road'
!add1.city := 'Kimmage'
!add1.state := 'Dublin'
!add1.zip := 'D15 K654'
!me.address := add1

!new ReceivablesAccount('meAcc')
!meAcc.number := me.id;
!insert(meAcc, me) into Has


-- Customer 'tom' initialization
!tom.id :='456456'
!tom.firstName := 'Thomas'
!tom.lastName :='Smith'
!tom.telHome := '0123456'
!tom.telWork := '6543210'

!add2.address1 := '25 garry st'
!add2.address2 := ' lake avenue'
!add2.city := 'grangegorman'
!add2.state := 'Dublin'
!add2.zip := 'D10 A546'
!tom.address := add2

!new ReceivablesAccount('tomAcc')
!tomAcc.number := tom.id;
!insert(tomAcc, tom) into Has


-- Customer 'jake' initialization
!jake.id :='789789'
!jake.firstName := 'Jacob'
!jake.lastName :='Lanert'
!jake.telHome := '0123456'
!jake.telWork := '6543210'

!add3.address1 := 'Houston'
!add3.address2 := ' Station'
!add3.city := 'Ohio'
!add3.state := 'Dublin'
```

```
!add3.zip := 'OH654'
!jake.address := add3

!new ReceivablesAccount('jakeAcc')
!jakeAcc.number := jake.id;
!insert(jakeAcc, jake) into Has


-- invoices
!arp.CreateInvoice('123789',20,meAcc)
!arp.CreateInvoice('651681',200,jakeAcc)
!arp.CreateInvoice('123789',159,meAcc)

--ARP operations
!arp.ProcessInvoices(ReceivablesAccount.allInstances())

!arp.CreateInvoice('010203',962,tomAcc)

!arp.ProcessInvoices(ReceivablesAccount.allInstances())

!arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
```