university of groningen

faculty of economics and business

2018011-EEF

Converting a Non-trivial Use Case into an SSD: An Exercise

August 2018

Bert de Brock

SOM is the research institute of the Faculty of Economics & Business at the University of Groningen. SOM has six programmes:

- Economics, Econometrics and Finance
- Global Economics & Management
- Innovation & Organization
- Marketing
- Operations Management & Operations Research
- Organizational Behaviour

Research Institute SOM
Faculty of Economics & Business
University of Groningen

Visiting address:
Nettelbosje 2
9747 AE Groningen
The Netherlands

Postal address:
P.O. Box 800
9700 AV Groningen
The Netherlands

T +31 50 363 7068/3815

www.rug.nl/feb/research

Converting a Non-trivial Use Case into an SSD: An Exercise

Bert de Brock
University of Groningen, Faculty of Economics and Business, Department of Economics, Econometrics and Finance
e.o.de.brock@rug.nl

Converting a non-trivial Use Case into an SSD: An exercise

Bert de Brock
Faculty of Economics and Business
University of Groningen
Groningen, The Netherlands
E.O.de.Brock@rug.nl

Abstract

In another paper we proposed a small but powerful grammar for specifying System Sequence Diagrams (SSDs). As an exercise, test, and illustration we want to apply it to a non-trivial, representative, extensive, and detailed real-life case study. The book *Applying UML and Patterns* of Larman contains such a case study. In the current technical report, we will convert his well-known use case 'Process Sale' into an SSD using our grammar.

Introduction

In [1] we proposed a grammar for describing system sequence diagrams (SSDs). As an exercise we want to convert the non-trivial use case 'Process Sale' in [2] into an SSD, including the extensions mentioned there. Throughout [2], Larman uses his NextGen Point-of-Sale system (POS) as an illustrative case study. In particular, he treats 'Process Sale' as a running example of a use case (UC). It is a very elaborated, representative real-life example. Although Larman treats this UC in depth, he only gives an SSD for the so called Main Success Scenario of the UC. We will now explain how to convert that UC into an SSD along the lines presented in [1]. The appendix contains the finally resulting SSD.

1. Main Success Scenario

We start with Larman's Main Success Scenario (MSS) of the UC, together with its corresponding SSD:

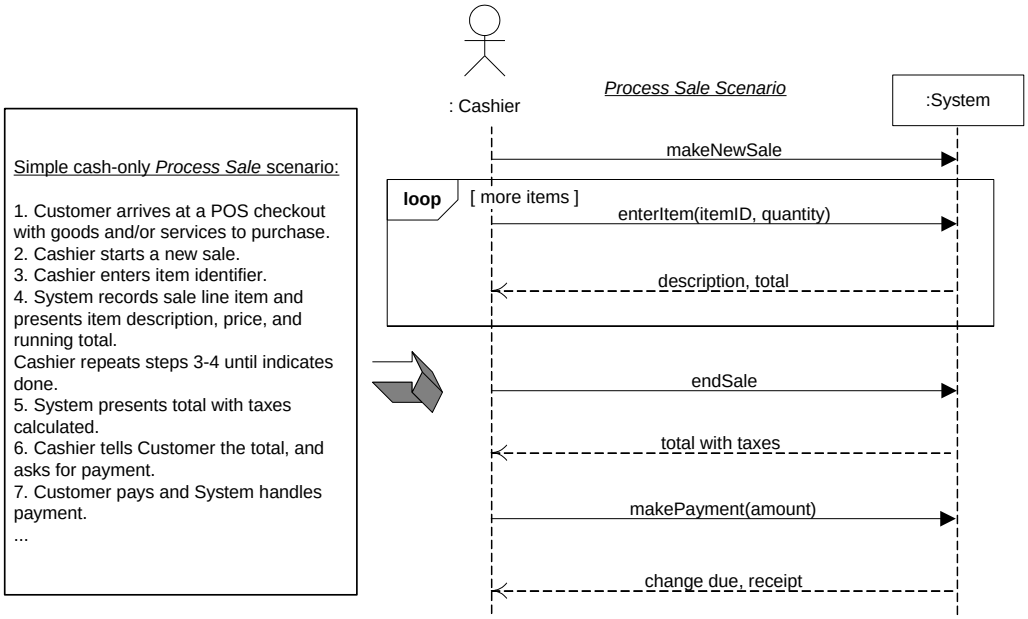

Figure 1: A use case and its corresponding SSD, taken from Section 10.5 of [2]

A few small remarks about the example itself:

- Input steps in the SSD (so steps of the form: Cashier ⟶ System) should indicate what *the system* has to do (e.g., 'makeNewSale'). Therefore, the name 'makePayment' in Figure 1 is misleading; it could better be something like 'handlePayment', for instance.
- Step 7 of the use case says: 'Customer pays and System handles payment'. We read this as 'Customer pays to the cashier, cashier requests the system to handle a cash payment, and System handles payment'. The first sub-step is not relevant for the SSD because there is no interaction with the system.
- In the last SSD-step, 'change due' shall be 'amount' – 'total' (from the previous SSD-steps).

- Note that the UC and SSD are not completely in line with each other (e.g., there is no 'quantity' in the UC and no 'price' in the SSD). When writing down the SSD along our lines, we will not follow the SSD of Larman, but his UC (as well as the step-numbering of his UC).

Some of his seven steps (viz. 4 and 7) actually consist of several steps and there is also an unnumbered step, just before Step 5. The order of steps 4.1 and 4.2 below is irrelevant. Some other steps (viz. 1 and 6) are not relevant for the SSD, because there is no interaction with the system. These steps are left out. When we apply our grammar for SSDs presented in [1], it results in the following SSD of the MSS:

2	Cashier ⟶ System **:** makeNewSale**;**
	for each item i of Customer
3	**do** Cashier ⟶ System **:** enterItem(itemID of i)**;**
4.1	System ⟶ System **:** recordSaleLine(i)**,** /* The comma indicates that the
4.2	System ⟶ Cashier **:** description of i, price of i, running total /* order of 4.1 and 4.2 is irrelevant
	end;
–	Cashier ⟶ System **:** endSale**;**
5	System ⟶ Cashier **:** total with taxes**;**
7.1	Cashier ⟶ System **:** handleCashPayment(amount)**;**
7.2	System ⟶ Cashier **:** change due, receipt

If we would like to simulate the 'life line' idea underlying the diagrams of UML (*User always left, below each other; System always right, also below each other*) then we could turn 'System ⟶ Cashier: …' into 'Cashier ⟵ System : …' and 'System ⟶ System : …' into ' System⇄ : …' (i.e., with 'System⇄' on the right-hand side). This would result in:

2	Cashier ⟶ System **:** makeNewSale**;**
	for each item i of Customer
3	**do** Cashier ⟶ System **:** enterItem(itemID of i)**;**
4.1	System⇄ **:** recordSaleLine(i)**;** /* The steps in a UML *sequence*
4.2	Cashier ⟵ System **:** description of i, price of i, running total /* diagram should be ordered
	end;
–	Cashier ⟶ System **:** endSale**;**
5	Cashier ⟵ System **:** total with taxes**;**
7.1	Cashier ⟶ System **:** handleCashPayment(amount)**;**
7.2	Cashier ⟵ System **:** change due, receipt

Somewhere else in [2] (in Section 6.8), the example also has steps 8, 9, and 10. In Step 10 there is no interaction with the system, so we ignore that step (just as steps 1 and 6). Step 9 is a separate step for presenting the receipt (so not as part of Step 7). Step 8 says: *System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).* We note that the order of these three steps is irrelevant, i.e., they can be applied in any order. We denote this 'freedom' as 'W1**,** W2**,** W3' meaning: 'W1, W2, and W3, in arbitrary order'. Hence, '**;**' is used when the order is relevant and '**,**' can be used to indicate that the order is irrelevant. When we name the external Accounting system AccSys and the external Inventory system InvSys, Step 8 consists of the following steps (in our notation):

8.1 System ⟶ System **:** log completed sale**,**
8.2 System ⟶ AccSys **:** sale and payment information**,**
8.3 System ⟶ InvSys **:** sale and payment information

An important difference between our SSDs and the classical UML-diagrams – such as the right side in Figure 1 – is (the scalability of) the layout, especially in the case of nested constructs. We can simply indicate the irrelevance of a certain order of steps (like between 8.1, 8.2, and 8.3). We introduce variables where relevant, e.g., a variable i in the loop above, so that we can clearly refer to it, e.g., indicate what is a property of i (e.g., description and price) and what isn't (e.g., running total).

2. Extensions

Besides the *Main Success Scenario* of the UC 'Process Sale', Larman also mentions various kinds of *extensions* of the UC. We will now work out almost all his extensions, towards SSD fragments. In the Appendix we present the finally resulting SSD. We will follow the order and numbering used by Larman ([2], Section 6.8).

*a) An important kind of extension is an interrupt ('Manager requests an override operation' in this case). An interrupt can occur at any moment, so anywhere in the UC/SSD. It can occur 0, 1, or more times. We have to specify each interrupt and how has to be handled. As described in [2], this example goes roughly as follows:

Manager ⟶ System**:** enterManagerMode**;**
System ⟶ System**:** change to Manager mode**;**
Manager ⟶ System**:** <some ManagerMode operation>**;**
System ⟶ System**:** change to Cashier mode

When we look back at this example, we might generalize it to a parameterized input changeModeTo(<role>). Also, it makes more sense that the manager him/herself indicates to go back to Cashier mode. This results in:

Manager ⟶ System**:** changeModeTo('Manager')**;**
System ⟶ System**:** change to mode 'Manager'**;**
Manager ⟶ System**:** <some ManagerMode operation>**;**
Manager ⟶ System**:** changeModeTo('Cashier')**;**
System ⟶ System**:** change to mode 'Cashier'

1a) Instead of starting a new sale (Step 2 in the UC, but actually the first step in the SSD), a suspended sale might be resumed. This starts as follows:

Cashier ⟶ System **:** resumeSuspendedSale(SaleID)**;**
System ⟶ Cashier **:** state of sale to be resumed**;**

after which the suspended sale can be continued. A scenario that Larman mentions is that the suspended sale might not be found by the system. So, those first steps should be:

Cashier ⟶ System **:** resumeSuspendedSale(SaleID)**;**
System ⟶ System **:** find sale with SaleID**;**
if sale is found
  then System ⟶ Cashier**:** state of sale to be resumed /* The suspended sale can be continued
  else System ⟶ Cashier**:** "Sale not found" /* The sale has to be re-entered as a new sale
end;

Our informal remarks (after '/*') could also be incorporated in the response of the system to the cashier.
In our opinion, (1a) is not an extension of the use case Process Sale but another use case ('Resume Sale').

2-4a) Another important kind of extension is the idea of an option: Once it *can* occur but it doesn't need to occur. In Larman's example, the customer might tell the cashier that (s)he has a tax-exempt status (e.g., being a senior). Similar to Extension *a, this is also an example of a 'free floating' extension: It can occur at any moment (until Step 5). The cashier verifies the status (a step outside the system), enters it, and the system records it. During the UC, this needs to be done at most once. It then leads to the following system interactions:

Cashier ⟶ System**:** enterStatusCode('tax-exempt')**;**
System ⟶ System**:** record Status Code 'tax-exempt'

3a) In Step 3, the entered item ID might be invalid, but if there is a human-readable item ID (e.g., a numeric UPC) then that item ID might be entered manually. Otherwise, if there is a price on the tag then that price might be entered manually. (As Larman describes, this should be done via an override by the manager, but for simplicity we neglect that.) Finally, the cashier can use the Help- or SearchItem-function or ask another cashier or the manager to find the correct item ID or price, and enter it. This is a subtly nested if-then-else construction:

Cashier ⟶ System**:** enterItem(itemID of i)**;** /* First try this (A)
System ⟶ System**:** check itemID of i**;**
if itemID of i is invalid
then System ⟶ Cashier**:** "Invalid item ID"**;**
  if there is a human-readable item ID /* check by Cashier
  then Cashier ⟶ System**:** enterManually(itemID of i) /* else try this (B)
  else if there is a price on the tag /* check by Cashier
    then Cashier ⟶ System**:** enterPrice(price of i)**;** /* else try this (C)
      Cashier ⟶ System**:** applyStandardTaxation
    else /* after finding the correct item ID or price /* else try either (B) or (C)

either Cashier ⟶ System**:** enterManually(itemID of i)
or Cashier ⟶ System**:** enterPrice(price of i)**;**
Cashier ⟶ System**:** applyStandardTaxation
end
end
end
end

With this construction, we force the intended order of preference: first try *enterItem*, else try *enterManually*, else try *enterPrice*, and then (after consultation) try either *enterManually* or *enterPrice*.

We recall an important difference between our SSDs and the classical UML-diagrams (as in the right-side of Figure 1): the scalability of the layout, especially in the case of such nested constructs.

3b) Sometimes a customer has multiple items of the same category (e.g., 5 bottles of the same wine). The cashier should then also be able to mention that quantity (instead of entering the item multiple times). We could change the original Step 3 into:

Cashier ⟶ System **:** enterItem(itemID of i [; q])

indicating that the parameter q (for *quantity*) is optional. Larman does not mention *quantity* in his UC but does mention it in his SSD; however, not as an option but as an obligation.

The default value of the parameter q should be 1. We could indicate that in the original Step 3 as:

Cashier ⟶ System**:** enterItem(itemID of i [; q **(default is** 1**)**])

This language construction is not in [1]. Extension 3b also influences Extension 3a.

3c) Some items require manual category and price entry (e.g., flowers). In that case, the step is something like:

Cashier ⟶ System **:** enterPricedItem(category code of i; price of i)

When we combine this with Extension 3b (multiple items of the same category), Step 3 could become:

if i is a special item /* check by Cashier
then Cashier ⟶ System **:** enterPricedItem(category code of i; price of i [; q **(default is** 1**)**])
else Cashier ⟶ System **:** enterItem(itemID of i [; q **(default is** 1**)**])
end

We now add our own extension: For some item types we must enter the weight (e.g., for fruit and vegetables). So, then we must distinguish three cases (and of course in time this could become more). Instead of a nested **if**-**then**-**else** we can use a kind of **case**-construction (similar to those in programming languages), as a generalization of the **if**-**then**-**else**. Our **case**-construction has the following general form:

case <expression> **is** <value> **then** <SSD>**,** <value> **then** <SSD>**,** … <value> **then** <SSD> [**else** <SSD>] **end**

Explanation: Depending on the value of the expression, a specific SSD will apply. There should be no duplicate values. If no value applies, the **else**-part applies, provided it is there (because the **else**-part is optional).

In the case of our extended example, the original Step 3 could now become as follows (with the (simple) text of the original Step 3 underlined):

case item type of i **is**
'priced' **then** Cashier ⟶ System **:** enterPricedItem(category code of i; price of i [; q **(default is** 1**)**])**,**
'weight' **then** Cashier ⟶ System **:** enterWeightItem(category code of i; weight of i [; q **(default is** 1**)**])
else Cashier ⟶ System **:** enterItem(itemID of i [; q **(default is** 1**)**])
end

By now, the advantages of starting with the (relatively simple) Main Success Scenario only, later followed by the stepwise (incremental or agile) development of a 'full' UC/SSD, might already become clear.

3-6a) The customer might ask the cashier to remove a certain item (in a certain quantity) from the purchase. This is another example of a 'free floating' extension. Moreover, the quantity parameter is optional, with default value 1. So, we could therefore define handleItemRemoval(i [; q **(default is** 1**)**]) as consisting of the steps

if quantity parameter is not given /* check by System
then System ⟶ System**:** let q be 1 **end;** /* First settle the optional parameter
Cashier ⟶ System**:** removeItem(itemID of i; q)**;**

System ⟶ System**:** remove item i in quantity q**;**
System ⟶ Cashier**:** new (running) total

But there is a complication: This can only be done by the cashier if the item value is less than the limit for cashiers to do it; otherwise, (the cashier knows that) the manager must do it. So, the second step might become:

if value of i ≤ cashier-limit /* check by Cashier (or System)
then Cashier ⟶ System **:** removeItem(itemID of i; q)
else Manager ⟶ System **:** removeItem(itemID of i; q)
end

3-6b) At any moment after Step 2 and before Step 7, the cashier *might* cancel the sale (e.g., on request of the customer). This can be done at most once during the use case. This interrupt has the following form:

Cashier ⟶ System**:** cancelSale**;**
System ⟶ System**:** delete sale**;**
System ⟶ Cashier**:** "Done"

3-6c) At any moment after Step 2 and before Step 7, the cashier *might* suspend the sale (e.g., on customer's request). During the use case this can be done at most once. This interrupt has the following form:

Cashier ⟶ System**:** suspendSale**;**
System ⟶ System**:** record suspended sale**;**
System ⟶ Cashier**:** suspend receipt (with sale ID and all the line items so far)

5b) A customer might say to be eligible for a discount (e.g., an employee or a preferred customer), in which case the cashier enters the customer identification and then the system presents the discount total. If this happens, it should happen just after Step 5. This option has the following form:

maybe begin Cashier ⟶ System**:** applyDiscount(Customer ID)**;**
System ⟶ System**:** apply discount to sale**;**
System ⟶ Cashier**:** new total with taxes
end

5c) A customer might have a credit on his/her account and might want to use it. In that case, the cashier enters the customer identification and then the system applies the credit (up to price = 0) and reduces the remaining credit. If this happens, it should happen just after Step 5 (and 5b). This option has the following form:

maybe begin Cashier ⟶ System**:** applyCredit(Customer ID)**;**
System ⟶ System**:** apply credit to sale**;**
System ⟶ System**:** reduce remaining credit**;**
System ⟶ Cashier**:** new total with taxes
end

7a-d) When the customer is about to pay (Step 7), there are several possibilities to pay: by cash, by credit, by check, by debit. Each of these possibilities require different steps. This typically asks for a **case**-construction. Larman works the cash and credit possibilities out in detail. The result is something like:

case payment method **is**
cash **then begin**
Cashier ⟶ System: makeCashPay(amount)**;**
System ⟶ Cashier: change due**;**
System ⟶ System: release cash drawer**;**
Cashier ⟶ System: ClosePayment**;**
System ⟶ System: record payment
end,
credit **then begin**
Customer ⟶ System: makeCreditPay(credit card; pin code)**;**
System ⟶ Cashier: payment info (for verification)**;**
Cashier ⟶ System: confirm**;**
System ⟶ AutSys: paymentApproval?**;** /* Request to Payment Authorisation system
:

:
end,
check **then** …**,**
debit **then** …
end

When we completely work out this instruction, it becomes large and hardly surveyable. Therefore, we would like to 'name' the instruction sequences after each **then**, for instance as follows:

define handleCashPayment **as**
begin
Cashier ⟶ System: makeCashPay(amount)**;**
System ⟶ Cashier: change due**;**
System ⟶ System: release cash drawer**;**
Cashier ⟶ System: ClosePayment**;**
System ⟶ System: record payment
end

We could do something similar for the other cases (credit, check, etc.), e.g.:

define handleCreditPayment **as**
begin
Customer ⟶ System: makeCreditPay(credit card; pin code)**;**
System ⟶ Cashier: payment info (for verification)**;**
Cashier ⟶ System: confirm**;**
System ⟶ AutSys: paymentApproval?**;** /* Request to Payment Authorisation system
:
:
end

Now we can rewrite the **case**-construction and introduce a general *handlePayment* as follows:

define handlePayment **as**
begin case payment method **is**
cash **then** handleCashPayment**,**
credit **then** handleCreditPayment**,**
check **then** handleCheckPayment**,**
debit **then** handleDebitPayment
end
end

In the future there might come more payment possibilities, and also some payment possibilities might disappear. This can all be managed here, in the **case**-construction within *handlePayment*, so at one spot.

7e) During the payment step, the cashier might cancel that step. The system then reverts to 'item entry' mode. Hence, the steps are:

Customer ⟶ System: cancelPayment**;**
System ⟶ System: revert to 'item entry' mode

We can consider this as an option *within* handlePayment.

7f) The customer *can* (but doesn't need to) present one or more coupons. So, this is another example of an option. It might go roughly as follows:

for each coupon c of the customer
do Cashier ⟶ System **:** recordCoupon(c)**;**
System ⟶ System **:** reduce price with value of c**;**
System ⟶ System **:** record usage of c**;**
System ⟶ Cashier **:** new total with taxes
end

If it happens, it should happen before handling the payment, so before Step 7. It might even happen before Extension 5c (using a credit). But it should take place after Extension 5b (applying a discount). The considered fragment of the SSD then becomes:

```
System ⟶ Cashier: total with taxes;                    /* Step 5
maybe handleDiscount;                                  /* Extension 5b
maybe handleCredit,                                    /* Extension 5c
maybe for each coupon c of the customer do … end;      /* Extension 7f
handlePayment;                                         /* Extensions 7a-d
```

9b) A customer might ask for a gift receipt, in which case the cashier asks the system for it and the system presents it. If this happens, it should happen at the end. This option has the following form:

maybe begin Cashier ⟶ System**:** giveGiftReceipt**;** System ⟶ Cashier**:** gift receipt **end**

9c) When the system wants to print a receipt (Step 9), it might detect that the printer ran out of paper. The system informs the cashier about it, the cashier replaces the paper (which is not an interaction with the system), and then the cashier requests the receipt. This option has the following form:

```
if printer is out of paper then                 /* check by System
    System ⟶ Cashier: "Out of paper";           /* Then the cashier must replace the paper
    Cashier ⟶ System: printReceipt
end
```

As we saw under extensions 3b, 3c, and 7f, for instance, extensions can influence each other. So, now and then we have to take other extensions into account as well. The finally resulting SSD for the 'fully dressed' version of Process Sale, including those influences, will be presented in the Appendix. We captured Process Sale in the form of a definition and named it *processSale* (so that it can easily be called upon).

References

[1] E.O. de Brock: Designing instruction languages, 2018, under review

[2] C. Larman: Applying UML and patterns, 3th edition, Addison Wesley Professional, 2005

Appendix: An SSD for the fully dressed use case Process Sale of Larman

We present an SSD that contains a representative subset of the extensions of the Process Sale example in Larman's Applying UML and patterns (see Section 6.8 of [2]). This SSD is the result of the discussion in our paper. But we start with our SSD for his *Main Success Scenario* of Process Sale, taking over the step numbers from that section:

Step 2	Cashier ⟶ System **:** makeNewSale**;**	
	for each item i of Customer	
Step 3	**do** Cashier ⟶ System **:** enterItem(itemID of i)**;**	
Step 4.1	System ⟶ System **:** recordSaleLine(i)**,**	
Step 4.2	System ⟶ Cashier **:** description of i, price of i, running total	/* price follows from price rules
	end;	
–	Cashier ⟶ System **:** endSale**;**	
Step 5	System ⟶ Cashier **:** total with taxes**;**	
Step 7	Cashier ⟶ System **:** handlePayment(amount)**;**	/* cash payment only
Step 8.1	System ⟶ System **:** log completed sale**,**	
Step 8.2	System ⟶ AccSys **:** sale and payment information**,**	/* external Accounting System
Step 8.3	System ⟶ InvSys **:** sale and payment information**;**	/* external Inventory System
Step 9	System ⟶ Cashier **:** receipt	

We capture our SSD for (a representative part of) Larman's 'fully dressed' version of Process Sale in the form of a definition, see below. We take over the step numbering and extension numbering used in [2], Section 6.8. Applying our grammar rules for SSDs as specified in [1], the *Main Scenario* (i.e., including the alternative flows) looks as follows:

DEFINE processSale **AS**
BEGIN

Cashier ⟶ System**:** makeNewSale**;**	/* Step 2
for each item i of Customer	/*
do handleItem(i [; q (**default is** 1)])**;**	/* Extensions 3b and 3c
System ⟶ System **:** recordSaleLine(i)**,**	/* Step 4.1
System ⟶ Cashier **:** description of i, price of i, and running total	/* Step 4.2
end;	/*
Cashier ⟶ System**:** endSale**;**	/* –
System ⟶ Cashier**:** total with taxes**;**	/* Step 5
maybe handleDiscount**;**	/* Extension 5b
maybe handleCredit**,**	/* Extension 5c
maybe handleCoupons**;**	/* Extension 7f
handlePayment**;**	/* Extensions 7a-d
System ⟶ System **:** log completed sale**,**	/* Step 8.1
System ⟶ AccSys **:** sale and payment information**,**	/* Step 8.2
System ⟶ InvSys **:** sale and payment information**;**	/* Step 8.3
if printer is out of paper **then** handlePaperShortage **end;**	/* Extension 9c
System ⟶ Cashier **:** receipt**;**	/* Step 9
maybe handleGiftReceipt**;**	/* Extension 9b
extra:	
handleManagerOverride**,**	/* Extension *a
handleTaxExempt **at most once,**	/* Extension 2-4a
handleItemRemoval(i [; q (**default is** 1)])**,**	/* Extension 3-6a
handleSaleCancellation **at most once,**	/* Extension 3-6b
handleSaleSuspension **at most once**	/* Extension 3-6c
end	

/* The Main Scenario refers to many definitions ('shorthands'), to be specified in a **with**-construction:

With

```
define handleItem(i [; q (default is 1) ]) as                          /* Extension 3b
begin  if second parameter is not given then System ⟶ System: let q be 1 end;   /* check by System
       case item type of i is
       'priced'  then Cashier ⟶ System : enterPricedItem(category code of i; price of i; q),   /*  Extension 3c+
       'weight' then Cashier ⟶ System : enterWeightItem(category code of i; weight of i; q)   /*  own extension
                 else  handleNormalItem(i; q)
       end
end;

define handleNormalItem(i; q) as
begin  Cashier ⟶ System: enterItem(itemID of i; q);                    /*  First try this        (A)
       System ⟶ System: check itemID of i;
       if itemID of i is invalid                                       /*  Extension 3a
       then System ⟶ Cashier: "Invalid item ID";
            if there is a human-readable item ID                       /*  check by Cashier
            then Cashier ⟶ System: enterManually(itemID of i; q)       /*  else try this         (B)
            else  if there is a price on the tag                       /*  check by Cashier
                  then Cashier ⟶ System: enterPrice(price of i; q);    /*  else try this         (C)
                       Cashier ⟶ System: applyStandardTaxation
                  else  /* after finding the correct item ID or price  /*  else try either (B) or (C)
                       either Cashier ⟶ System: enterManually(itemID of i; q)
                       or  Cashier ⟶ System: enterPrice(price of i; q);
                           Cashier ⟶ System: applyStandardTaxation
                       end
                  end
            end
       end
end;

define handleDiscount as                                               /*  Extension 5b
begin  Cashier ⟶ System: applyDiscount(Customer ID);
       System ⟶ System: apply discount to sale;
       System ⟶ Cashier: new total with taxes
end;

define handleCredit as                                                 /*  Extension 5c
begin  Cashier  ⟶ System: applyCredit(Customer ID);
       System  ⟶ System: apply credit to sale;
       System  ⟶ System: reduce remaining credit;
       System  ⟶ Cashier: new total with taxes
end;

define handleCoupons as                                                /*  Extension 7f
begin  for each coupon c of the customer
       do  Cashier  ⟶  System: recordCoupon(c);
           System  ⟶  System: reduce price with value of c;
           System  ⟶  System: record usage of c;
           System  ⟶  Cashier: new total with taxes
       end
end;

define handlePayment as
begin  case payment method is
       cash    then handleCashPayment,                                 /*  Extension 7a
       credit  then handleCreditPayment,                               /*  Extension 7b
       check   then handleCheckPayment,                                /*  Extension 7c
       debit   then handleDebitPayment                                 /*  Extension 7d
       end;
extra: handlePaymentCancellation end                                   /*  Extension 7e
end;
```

```
define handleCashPayment as                                    /* Extension 7a
begin  Cashier ⟶ System: makeCashPay(amount);
       System ⟶ Cashier: change due;
       System ⟶ System: release cash drawer;
       Cashier ⟶ System: ClosePayment;
       System ⟶ System: record payment
end;

define handleCreditPayment as                                  /* Extension 7b
begin  Customer ⟶ System: makeCreditPay(credit card; pin code);
       System   ⟶ Cashier: payment info (for verification);
       Cashier  ⟶ System: confirm;
       System   ⟶ AutSys: paymentApproval?;          /* Request to the Payment Authorisation system
       :
       :
end;

define handleCheckPayment as                                   /* Extension 7c
begin  :
       :
       :
end;

define handleDebitPayment as                                   /* Extension 7d
begin  :
       :
       :
end;

define handlePaymentCancellation as                            /* Extension 7e
begin  Customer ⟶ System: cancelPayment;
       System   ⟶ System: revert to 'item entry' mode
end;

define handlePaperShortage as                                  /* Extension 9c
begin  System ⟶ Cashier: "Out of paper";                       /* Then the cashier must replace the paper
       Cashier ⟶ System: printReceipt
end;

define handleGiftReceipt as                                    /* Extension 9b
begin  Cashier ⟶ System: giveGiftReceipt;
       System ⟶ Cashier: gift receipt
end;

define handleManagerOverride as                                /* Extension *a
begin  Manager ⟶ System: changeModeTo('Manager');
       System  ⟶ System: change to mode 'Manager';
       Manager ⟶ System: <some ManagerMode operation>;
       Manager ⟶ System: changeModeTo('Cashier');
       System  ⟶ System: change to mode 'Cashier'
end;

define handleTaxExempt as                                      /* Extension 2-4a
begin  Cashier ⟶ System: enterStatusCode('tax-exempt');
       System  ⟶ System: record Status Code 'tax-exempt'
end;

define handleItemRemoval(i [; q (default is 1) ]) as           /* Extension 3-6a
begin  if second parameter is not given
       then System ⟶ System: let q be 1 end;
       if value of i ≤ cashier-limit
         then Cashier  ⟶ System: removeItem(itemID of i; q)
         else Manager  ⟶ System: removeItem(itemID of i; q)
```

end;
System ⟶ System**:** remove item i in quantity q**;**
System ⟶ Cashier**:** new (running) total
end;

define handleSaleCancellation **as** /* Extension 3-6b
begin Cashier ⟶ System**:** cancelSale**;**
System ⟶ System**:** delete sale**;**
System ⟶ Cashier**:** “Done”
end;

define handleSaleSuspension **as** /* Extension 3-6c
begin Cashier ⟶ System**:** suspendSale**;**
System ⟶ System**:** record suspended sale**;**
System ⟶ Cashier**:** suspend receipt (with sale ID and all the line items so far)
end

End

END

List of research reports

13001-EEF: Kuper, G.H. and M. Mulder, Cross-border infrastructure constraints, regulatory measures and economic integration of the Dutch – German gas market

13002-EEF: Klein Goldewijk, G.M. and J.P.A.M. Jacobs, The relation between stature and long bone length in the Roman Empire

13003-EEF: Mulder, M. and L. Schoonbeek, Decomposing changes in competition in the Dutch electricity market through the Residual Supply Index

13004-EEF: Kuper, G.H. and M. Mulder, Cross-border constraints, institutional changes and integration of the Dutch – German gas market

13005-EEF: Wiese, R., Do political or economic factors drive healthcare financing privatisations? Empirical evidence from OECD countries

13006-EEF: Elhorst, J.P., P. Heijnen, A. Samarina and J.P.A.M. Jacobs, State transfers at different moments in time: A spatial probit approach

13007-EEF: Mierau, J.O., The activity and lethality of militant groups: Ideology, capacity, and environment

13008-EEF: Dijkstra, P.T., M.A. Haan and M. Mulder, The effect of industry structure and yardstick design on strategic behavior with yardstick competition: an experimental study

13009-GEM: Hoorn, A.A.J. van, Values of financial services professionals and the global financial crisis as a crisis of ethics

13010-EEF: Boonman, T.M., Sovereign defaults, business cycles and economic growth in Latin America, 1870-2012

13011-EEF: He, X., J.P.A.M Jacobs, G.H. Kuper and J.E. Ligthart, On the impact of the global financial crisis on the euro area

13012-GEM: Hoorn, A.A.J. van, Generational shifts in managerial values and the coming of a global business culture

13013-EEF: Samarina, A. and J.E. Sturm, Factors leading to inflation targeting – The impact of adoption

13014-EEF: Allers, M.A. and E. Merkus, Soft budget constraint but no moral hazard? The Dutch local government bailout puzzle

13015-GEM: Hoorn, A.A.J. van, Trust and management: Explaining cross-national differences in work autonomy

13016-EEF: Boonman, T.M., J.P.A.M. Jacobs and G.H. Kuper, Sovereign debt crises in Latin America: A market pressure approach

13017-GEM: Oosterhaven, J., M.C. Bouwmeester and M. Nozaki, The impact of production and infrastructure shocks: A non-linear input-output programming approach, tested on an hypothetical economy

13018-EEF: Cavapozzi, D., W. Han and R. Miniaci, Alternative weighting structures for multidimensional poverty assessment

14001-OPERA: Germs, R. and N.D. van Foreest, Optimal control of production-inventory systems with constant and compound poisson demand

14002-EEF: Bao, T. and J. Duffy, Adaptive vs. eductive learning: Theory and evidence

14003-OPERA: Syntetos, A.A. and R.H. Teunter, On the calculation of safety stocks

14004-EEF: Bouwmeester, M.C., J. Oosterhaven and J.M. Rueda-Cantuche, Measuring the EU value added embodied in EU foreign exports by consolidating 27 national supply and use tables for 2000-2007

14005-OPERA: Prak, D.R.J., R.H. Teunter and J. Riezebos, Periodic review and continuous ordering

14006-EEF: Reijnders, L.S.M., The college gender gap reversal: Insights from a life-cycle perspective

14007-EEF: Reijnders, L.S.M., Child care subsidies with endogenous education and fertility

14008-EEF: Otter, P.W., J.P.A.M. Jacobs and A.H.J. den Reijer, A criterion for the number of factors in a data-rich environment

14009-EEF: Mierau, J.O. and E. Suari Andreu, Fiscal rules and government size in the European Union

14010-EEF: Dijkstra, P.T., M.A. Haan and M. Mulder, Industry structure and collusion with uniform yardstick competition: theory and experiments

14011-EEF: Huizingh, E. and M. Mulder, Effectiveness of regulatory interventions on firm behavior: a randomized field experiment with e-commerce firms

14012-GEM: Bressand, A., Proving the old spell wrong: New African hydrocarbon producers and the ‘resource curse’

14013-EEF: Dijkstra P.T., Price leadership and unequal market sharing: Collusion in experimental markets

14014-EEF: Angelini, V., M. Bertoni, and L. Corazzini, Unpacking the determinants of life satisfaction: A survey experiment

14015-EEF: Heijdra, B.J., J.O. Mierau, and T. Trimborn, Stimulating annuity markets

14016-GEM: Bezemer, D., M. Grydaki, and L. Zhang, Is financial development bad for growth?

14017-EEF: De Cao, E. and C. Lutz, Sensitive survey questions: measuring attitudes regarding female circumcision through a list experiment

14018-EEF: De Cao, E., The height production function from birth to maturity

14019-EEF: Allers, M.A. and J.B. Geertsema, The effects of local government amalgamation on public spending and service levels. Evidence from 15 years of municipal boundary reform

14020-EEF: Kuper, G.H. and J.H. Veurink, Central bank independence and political pressure in the Greenspan era

14021-GEM: Samarina, A. and D. Bezemer, Do Capital Flows Change Domestic Credit Allocation?

14022-EEF: Soetevent, A.R. and L. Zhou, Loss Modification Incentives for Insurers Under ExpectedUtility and Loss Aversion

14023-EEF: Allers, M.A. and W. Vermeulen, Fiscal Equalization, Capitalization and the Flypaper Effect.

14024-GEM: Hoorn, A.A.J. van, Trust, Workplace Organization, and Comparative Economic Development.

14025-GEM: Bezemer, D., and L. Zhang, From Boom to Bust in de Credit Cycle: The Role of Mortgage Credit.

14026-GEM: Zhang, L., and D. Bezemer, How the Credit Cycle Affects Growth: The Role of Bank Balance Sheets.

14027-EEF: Bružikas, T., and A.R. Soetevent, Detailed Data and Changes in Market Structure: The Move to Unmanned Gasoline Service Stations.

14028-EEF: Bouwmeester, M.C., and B. Scholtens, Cross-border Spillovers from European Gas Infrastructure Investments.

14029-EEF: Lestano, and G.H. Kuper, Correlation Dynamics in East Asian Financial Markets.

14030-GEM: Bezemer, D.J., and M. Grydaki, Nonfinancial Sectors Debt and the U.S. Great Moderation.

14031-EEF: Hermes, N., and R. Lensink, Financial Liberalization and Capital Flight: Evidence from the African Continent.

14032-OPERA: Blok, C. de, A. Seepma, I. Roukema, D.P. van Donk, B. Keulen, and R. Otte, Digitalisering in Strafrechtketens: Ervaringen in Denemarken, Engeland, Oostenrijk en Estland vanuit een Supply Chain Perspectief.

14033-OPERA: Olde Keizer, M.C.A., and R.H. Teunter, Opportunistic condition-based maintenance and aperiodic inspections for a two-unit series system.

14034-EEF: Kuper, G.H., G. Sierksma, and F.C.R. Spieksma, Using Tennis Rankings to Predict Performance in Upcoming Tournaments

15001-EEF: Bao, T., X. Tian, X. Yu, Dictator Game with Indivisibility of Money

15002-GEM: Chen, Q., E. Dietzenbacher, and B. Los, The Effects of Ageing and Urbanization on China's Future Population and Labor Force

15003-EEF: Allers, M., B. van Ommeren, and B. Geertsema, Does intermunicipal cooperation create inefficiency? A comparison of interest rates paid by intermunicipal organizations, amalgamated municipalities and not recently amalgamated municipalities

15004-EEF: Dijkstra, P.T., M.A. Haan, and M. Mulder, Design of Yardstick Competition and Consumer Prices: Experimental Evidence

15005-EEF: Dijkstra, P.T., Price Leadership and Unequal Market Sharing: Collusion in Experimental Markets

15006-EEF: Anufriev, M., T. Bao, A. Sutin, and J. Tuinstra, Fee Structure, Return Chasing and Mutual Fund Choice: An Experiment

15007-EEF: Lamers, M., Depositor Discipline and Bank Failures in Local Markets During the Financial Crisis

15008-EEF: Oosterhaven, J., On de Doubtful Usability of the Inoperability IO Model

15009-GEM: Zhang, L. and D. Bezemer, A Global House of Debt Effect? Mortgages and Post-Crisis Recessions in Fifty Economies

15010-I&O: Hooghiemstra, R., N. Hermes, L. Oxelheim, and T. Randøy, The Impact of Board Internationalization on Earnings Management

15011-EEF: Haan, M.A., and W.H. Siekman, Winning Back the Unfaithful while Exploiting the Loyal: Retention Offers and Heterogeneous Switching Costs

15012-EEF: Haan, M.A., J.L. Moraga-González, and V. Petrikaite, Price and Match-Value Advertising with Directed Consumer Search

15013-EEF: Wiese, R., and S. Eriksen, Do Healthcare Financing Privatisations Curb Total Healthcare Expenditures? Evidence from OECD Countries

15014-EEF: Siekman, W.H., Directed Consumer Search

15015-GEM: Hoorn, A.A.J. van, Organizational Culture in the Financial Sector: Evidence from a Cross-Industry Analysis of Employee Personal Values and Career Success

15016-EEF: Te Bao, and C. Hommes, When Speculators Meet Constructors: Positive and Negative Feedback in Experimental Housing Markets

15017-EEF: Te Bao, and Xiaohua Yu, Memory and Discounting: Theory and Evidence

15018-EEF: Suari-Andreu, E., The Effect of House Price Changes on Household Saving Behaviour: A Theoretical and Empirical Study of the Dutch Case

15019-EEF: Bijlsma, M., J. Boone, and G. Zwart, Community Rating in Health Insurance: Trade-off between Coverage and Selection

15020-EEF: Mulder, M., and B. Scholtens, A Plant-level Analysis of the Spill-over Effects of the German *Energiewende*

15021-GEM: Samarina, A., L. Zhang, and D. Bezemer, Mortgages and Credit Cycle Divergence in Eurozone Economies

16001-GEM: Hoorn, A. van, How Are Migrant Employees Manages? An Integrated Analysis

16002-EEF: Soetevent, A.R., Te Bao, A.L. Schippers, A Commercial Gift for Charity

16003-GEM: Bouwmeerster, M.C., and J. Oosterhaven, Economic Impacts of Natural Gas Flow Disruptions

16004-MARK: Holtrop, N., J.E. Wieringa, M.J. Gijsenberg, and P. Stern, Competitive Reactions to Personal Selling: The Difference between Strategic and Tactical Actions

16005-EEF: Plantinga, A. and B. Scholtens, The Financial Impact of Divestment from Fossil Fuels

16006-GEM: Hoorn, A. van, Trust and Signals in Workplace Organization: Evidence from Job Autonomy Differentials between Immigrant Groups

16007-EEF: Willems, B. and G. Zwart, Regulatory Holidays and Optimal Network Expansion

16008-GEF: Hoorn, A. van, Reliability and Validity of the Happiness Approach to Measuring Preferences

16009-EEF: Hinloopen, J., and A.R. Soetevent, (Non-)Insurance Markets, Loss Size Manipulation and Competition: Experimental Evidence

16010-EEF: Bekker, P.A., A Generalized Dynamic Arbitrage Free Yield Model

16011-EEF: Mierau, J.A., and M. Mink, A Descriptive Model of Banking and Aggregate Demand

16012-EEF: Mulder, M. and B. Willems, Competition in Retail Electricity Markets: An Assessment of Ten Year Dutch Experience

16013-GEM: Rozite, K., D.J. Bezemer, and J.P.A.M. Jacobs, Towards a Financial Cycle for the US, 1873-2014

16014-EEF: Neuteleers, S., M. Mulder, and F. Hindriks, Assessing Fairness of Dynamic Grid Tariffs

16015-EEF: Soetevent, A.R., and T. Bružikas, Risk and Loss Aversion, Price Uncertainty and the Implications for Consumer Search

16016-HRM&OB: Meer, P.H. van der, and R. Wielers, Happiness, Unemployment and Self-esteem

16017-EEF: Mulder, M., and M. Pangan, Influence of Environmental Policy and Market Forces on Coal-fired Power Plants: Evidence on the Dutch Market over 2006-2014

16018-EEF: Zeng,Y., and M. Mulder, Exploring Interaction Effects of Climate Policies: A Model Analysis of the Power Market

16019-EEF: Ma, Yiqun, Demand Response Potential of Electricity End-users Facing Real Time Pricing

16020-GEM: Bezemer, D., and A. Samarina, Debt Shift, Financial Development and Income Inequality in Europe

16021-EEF: Elkhuizen, L, N. Hermes, and J. Jacobs, Financial Development, Financial Liberalization and Social Capital

16022-GEM: Gerritse, M., Does Trade Cause Institutional Change? Evidence from Countries South of the Suez Canal

16023-EEF: Rook, M., and M. Mulder, Implicit Premiums in Renewable-Energy Support Schemes

17001-EEF: Trinks, A., B. Scholtens, M. Mulder, and L. Dam, Divesting Fossil Fuels: The Implications for Investment Portfolios

17002-EEF: Angelini, V., and J.O. Mierau, Late-life Health Effects of Teenage Motherhood

17003-EEF: Jong-A-Pin, R., M. Laméris, and H. Garretsen, Political Preferences of (Un)happy Voters: Evidence Based on New Ideological Measures

17004-EEF: Jiang, X., N. Hermes, and A. Meesters, Financial Liberalization, the Institutional Environment and Bank Efficiency

17005-EEF: Kwaak, C. van der, Financial Fragility and Unconventional Central Bank Lending Operations

17006-EEF: Postelnicu, L. and N. Hermes, The Economic Value of Social Capital

17007-EEF: Ommeren, B.J.F. van, M.A. Allers, and M.H. Vellekoop, Choosing the Optimal Moment to Arrange a Loan

17008-EEF: Bekker, P.A., and K.E. Bouwman, A Unified Approach to Dynamic Mean-Variance Analysis in Discrete and Continuous Time

17009-EEF: Bekker, P.A., Interpretable Parsimonious Arbitrage-free Modeling of the Yield Curve

17010-GEM: Schasfoort, J., A. Godin, D. Bezemer, A. Caiani, and S. Kinsella, Monetary Policy Transmission in a Macroeconomic Agent-Based Model

17011-I&O: Bogt, H. ter, Accountability, Transparency and Control of Outsourced Public Sector Activities

17012-GEM: Bezemer, D., A. Samarina, and L. Zhang, The Shift in Bank Credit Allocation: New Data and New Findings

17013-EEF: Boer, W.I.J. de, R.H. Koning, and J.O. Mierau, Ex-ante and Ex-post Willingness-to-pay for Hosting a Major Cycling Event

17014-OPERA: Laan, N. van der, W. Romeijnders, and M.H. van der Vlerk, Higher-order Total Variation Bounds for Expectations of Periodic Functions and Simple Integer Recourse Approximations

17015-GEM: Oosterhaven, J., Key Sector Analysis: A Note on the Other Side of the Coin

17016-EEF: Romensen, G.J., A.R. Soetevent: Tailored Feedback and Worker Green Behavior: Field Evidence from Bus Drivers

17017-EEF: Trinks, A., G. Ibikunle, M. Mulder, and B. Scholtens, Greenhouse Gas Emissions Intensity and the Cost of Capital

17018-GEM: Qian, X. and A. Steiner, The Reinforcement Effect of International Reserves for Financial Stability

17019-GEM/EEF: Klasing, M.J. and P. Milionis, The International Epidemiological Transition and the Education Gender Gap

2018001-EEF: Keller, J.T., G.H. Kuper, and M. Mulder, Mergers of Gas Markets Areas and Competition amongst Transmission System Operators: Evidence on Booking Behaviour in the German Markets

2018002-EEF: Soetevent, A.R. and S. Adikyan, The Impact of Short-Term Goals on Long-Term Objectives: Evidence from Running Data

2018003-MARK: Gijsenberg, M.J. and P.C. Verhoef, Moving Forward: The Role of Marketing in Fostering Public Transport Usage

2018004-MARK: Gijsenberg, M.J. and V.R. Nijs, Advertising Timing: In-Phase or Out-of-Phase with Competitors?

2018005-EEF: Hulshof, D., C. Jepma, and M. Mulder, Performance of Markets for European Renewable Energy Certificates

2018006-EEF: Fosgaard, T.R., and A.R. Soetevent, Promises Undone: How Committed Pledges Impact Donations to Charity

2018007-EEF: Durán, N. and J.P. Elhorst, A Spatio-temporal-similarity and Common Factor Approach of Individual Housing Prices: The Impact of Many Small Earthquakes in the North of Netherlands

2018008-EEF: Hermes, N., and M. Hudon, Determinants of the Performance of Microfinance Institutions: A Systematic Review

2018009-EEF: Katz, M., and C. van der Kwaak, The Macroeconomic Effectiveness of Bank Bail-ins

2018010-OPERA: Prak, D., R.H. Teunter, M.Z. Babai, A.A. Syntetos, and J.E. Boylan, Forecasting and Inventory Control with Compound Poisson Demand Using Periodic Demand Data

2018011-EEF: Brock, B. de, Converting a Non-trivial Use Case into an SSD: An Exercise

www.rug.nl/feb