**PlantUML-code used in the book**

**Developing Information Systems Accurately:**

A Wholistic Approach

**Table of contents**

[Preface 3](#_Toc110627961)

[2 Developing a Functional Requirement 4](#_Toc110627962)

[2.3 User Stories 4](#_Toc110627963)

[2.8.2 Generating Graphical SSDs from Textual SSDs 4](#_Toc110627964)

[2.9 Overview of Our Approach to Develop a User Wish 9](#_Toc110627965)

[4 Domain Modelling 10](#_Toc110627966)

[4.1 Domain Models 10](#_Toc110627967)

[4.2 Finding/Discovering/Determining Relevant Candidates 11](#_Toc110627968)

[4.3.1 Many-to-Many Associations 12](#_Toc110627969)

[4.3.2 Ternary Associations and Beyond 12](#_Toc110627970)

[4.3.3 Individual Items Versus ‘Catalogue’ Items 13](#_Toc110627971)

[4.3.4 Directed Graphs 14](#_Toc110627972)

[4.3.5 Trees and the Like 15](#_Toc110627973)

[4.3.6 Other Concepts Related to Themselves 16](#_Toc110627974)

[4.3.7 Generalization and Specialization 17](#_Toc110627975)

[4.4 Summary 18](#_Toc110627976)

[5 Conceptual Data Models 19](#_Toc110627977)

[5.2 From *Domain Models* to *Conceptual Data Models*: An Example 19](#_Toc110627978)

[5.3 From *Domain Models* to *Conceptual Data Models*: General Case 22](#_Toc110627979)

[5.4 Using Arrows for Conceptual Data Models 23](#_Toc110627980)

[5.6.1 Transforming Many-to-Many Associations 26](#_Toc110627981)

[5.6.2 Treating Associations in General 26](#_Toc110627982)

[5.6.3 Individual Items Versus ‘Catalogue’ Items 28](#_Toc110627983)

[5.6.4 Directed Graphs 29](#_Toc110627984)

[5.6.5 Trees and the Like 31](#_Toc110627985)

[5.6.6 Other Concepts Related to Themselves 32](#_Toc110627986)

[5.6.7 Generalization and Specialization 33](#_Toc110627987)

[6 Directions for Implementation 34](#_Toc110627988)

[6.3 Interaction with a (Software) System 34](#_Toc110627989)

[6.4.1 From *Conceptual Data Models* to *Data Models in SQL* 35](#_Toc110627990)

[6.5 Generating an Initial Class Diagram 37](#_Toc110627991)

[7 Organizing and Managing the Development Process 38](#_Toc110627992)

[7.2.1 Waterfall Methods 38](#_Toc110627993)

[7.2.2 Parallel Development 38](#_Toc110627994)

[7.2.4 Prototyping 39](#_Toc110627995)

[7.2.5 Iterative Development and Incremental Development 40](#_Toc110627996)

[8 A Non-Trivial University Example Worked out 41](#_Toc110627997)

[8.3 A Simple Domain Model for the Situation 41](#_Toc110627998)

[8.4 From Domain Model to an Initial Conceptual Data Model 41](#_Toc110627999)

[8.5 A Refined Conceptual Data Model Including References 42](#_Toc110628000)

[8.7 Conceptual Data Model with ‘all’ Relevant Properties 44](#_Toc110628001)

[8.9.1 Informally 46](#_Toc110628002)

[8.11 Specification in SQL 47](#_Toc110628003)

[9 Converting a Large Use Case 49](#_Toc110628004)

[9.3 Converting a Large Textual SSD to a Graphical SSD 49](#_Toc110628005)

[10 Development Example where Requirements Constantly Change 55](#_Toc110628006)

[10.4 Variable Thresholds per Room Type 55](#_Toc110628007)

[10.6 A Corresponding Textual SSD and Graphical SSD 56](#_Toc110628008)

[10.11 Interactions Between Our System and its Environment 57](#_Toc110628009)

[10.12 Looking Back: Typical Ingredients of an IS and a Control System 57](#_Toc110628010)

[Appendix: Our Plantuml Tutorial 58](#_Toc110628011)

[A.1 From Textual SSDs to Graphical SSDs 58](#_Toc110628012)

[A.2 Domain Models 58](#_Toc110628013)

[A.3 Conceptual Data Models 58](#_Toc110628014)

[A.4 Class Diagrams 58](#_Toc110628015)

[A.5 System as Interface + Kernel 58](#_Toc110628016)

[A.6 Use Case Diagrams 58](#_Toc110628017)

On the next pages, each text from ‘**@startuml**’ up to ‘**@enduml**’ can be substituted in <http://www.plantuml.com/plantuml/uml/SyfFKj2rKt3CoKnELR1Io4ZDoSa70000>

# Preface



@startuml

hide circle

hide members

allowmixing

Package “\nProblem \n Space\n” <<Cloud>> {

}

database “\nSystem\n\n”

“\nProblem \n Space\n” .> “\nConceptual \nModel\n”: ??

“\nConceptual \nModel\n” -> “\nSystem\n\n”: !!

@enduml

# 2 Developing a Functional Requirement

## 2.3 User Stories

**Figure 2.2**

@startuml

left to right direction

: Program secretary : --> (Create a Course Registration)

: Program director : --> (Create a Course)

: Program director : --> (Retrieve the set of my Courses)

: Program director : --> (Retrieve the Registrations for a Course)

: Program secretary : --> (Retrieve the Registrations for a Course)

: Program director : --> (Update a Course name)

: Program director : --> (Delete a Course)

: Program secretary : --> (Delete a Course Registration)

: Sysadmin : --> (Initialize the state)

@enduml

### 2.8.2 Generating Graphical SSDs from Textual SSDs

**Table 2.7:** Ai **🠮** Aj: Message (if Ai occurred earlier than Aj)

@startuml

participant Ai

participant "...." #white

participant Aj

Ai -> Aj: Message

@enduml

**Table 2.7:** Ai **🠮** Aj: Message (if Ai occurred later than Aj)

@startuml

participant Aj

participant "...." #white

participant Ai

Ai -> Aj: Message

@enduml

**Table 2.7:** Ai **🠮** Ai**:** Message

@startuml

Ai -> Ai: Message

@enduml

**Table 2.7:** S1**;** S2

@startuml

participant Ai1

participant Ai2

participant Aj1

note over Ai1, Aj1: //D//(S1)

note over Ai2, Aj2: //D//(S2)

@enduml

**Table 2.7:** **if** C **then** S **end**

@startuml

participant Ai

participant "...." #white

participant Aj

group if [ C ]

note over Ai, Aj: //D//(S)

end

@enduml

**Table 2.7:** **if** C **then** S1 **else** S2 **end**

@startuml

participant Ai1

participant Ai2

participant Aj1

group if [ C ]

note over Ai1, Aj1: //D//(S1)

else else

note over Ai2, Aj2: //D//(S2)

end

@enduml

**Table 2.7:** **if** A: C **then** S **end**

@startuml

participant Ai

participant "...." #white

participant Aj

group if [ A: C ]

note over Ai, Aj: //D//(S)

end

@enduml

**Table 2.7:** **if** A: C **then** S1 **else** S2 **end**

@startuml

participant Ai1

participant Ai2

participant Aj1

group if [ A: C ]

note over Ai1, Aj1: //D//(S1)

else else /\* not C \*/

note over Ai2, Aj2: //D//(S2)

end

@enduml

**Table 2.7:** **while** C **do** S **end**

@startuml

participant Ai

participant "...." #white

participant Aj

group while [ C ]

note over Ai, Aj: //D//(S)

end

@enduml

**Table 2.7:** **repeat** S **until** C

@startuml

participant Ai

participant "...." #white

participant Aj

group repeat

note over Ai, Aj: //D//(S)

rnote over Ai, Aj #white:\*\*until\*\* C

end

@enduml

**Table 2.7:** **while** A: C **do** S **end**

@startuml

participant Ai

participant "...." #white

participant Aj

group while [ A: C ]

note over Ai, Aj: //D//(S)

end

@enduml

**Table 2.7:** **repeat** S **until** A: C

@startuml

participant Ai

participant "...." #white

participant Aj

group repeat

note over Ai, Aj: //D//(S)

rnote over Ai, Aj #white: \*\*until\*\* A: C

end

@enduml

**Table 2.7:** **for each** E **do** S **end**

@startuml

participant Ai

participant "...." #white

participant Aj

group for each [ E ]

note over Ai, Aj: //D//(S)

end

@enduml

**Table 2.7:** S1**,** S2

@startuml

participant Ai1

participant Ai2

participant Aj1

group in any order

note over Ai1, Aj1: //D//(S1)

note over Ai2, Aj2: //D//(S2)

end

@enduml

**Table 2.7:** **maybe** S **end**

@startuml

participant Ai

participant "...." #white

participant Aj

group maybe

note over Ai, Aj: //D//(S)

end

@enduml

**Table 2.7:** **either** S1 **or** S2 **end**

@startuml

participant Ai1

participant Ai2

participant Aj1

group either

note over Ai1, Aj1: //D//(S1)

else or

note over Ai2, Aj2: //D//(S2)

end

@enduml

**Table 2.7:** **begin** S **end**

@startuml

participant Ai

participant "...." #white

participant Aj

group block

note over Ai, Aj: //D//(S)

end

@enduml

**Table 2.7:** **perform** N

@startuml

participant Ai

participant "...." #white

participant Aj

group perform

rnote over Ai, Aj #white: N

end

@enduml

**Table 2.7:** **define** N **as** S **end**

@startuml

title N \n

participant Ai

participant "...." #white

participant Aj

note over Ai, Aj: //D//(S)

@enduml

**Figure 2.3**

@startuml

title <size:18> Process a Sale </size> \n\n

Customer -> Customer: arrive at checkout \nwith items to purchase

Cashier -> System: StartNewSale

group repeat

 Cashier -> System: EnterItem(<item identifier>)

group in any order

 System -> System: RecordSaleLine(<item identifier>)

 System -> Cashier: description, price, \nand running total

end

rnote over Cashier #white:\*\*until\*\* cashier is done

end

Cashier -> System: EndSale

System -> Cashier: total with taxes

Cashier -> Customer: total

Cashier -> Customer: request for payment

Customer -> System: process payment

System -> System: handle payment

group in any order

System -> System: log completed sale

System -> AccSys: sale and payment info

System -> InvSys: sale and payment info

end

System -> Cashier: receipt

Customer -> Customer: leave with receipt and goods

@enduml

**Figure 2.4**

@startuml

note over User, System: //D//(A)

group if [ H2 ]

note over User, System: //D//(A2)

end

group if [ H1 ]

note over User, System: //D//(B2)

else else

group if [ H3 ]

note over User, System: //D//(B3)

else else

note over User, System: //D//(B)

end

end

group either

note over User, System: //D//(C)

else or

note over User, System: //D//(C2)

else or

note over User, System: //D//(C3)

end

note over User, System: //D//(D)

group maybe

group in any order

note over User, System: //D//(E1)

note over User, System: //D//(E2)

end

end

@enduml

## 2.9 Overview of Our Approach to Develop a User Wish

**Figure 2.7**

@startuml

User -> System: <pUW>

group if [student is a foreigner]

 rnote over User, System #white: …..

else else

group if [(s)he was not a student before]

 System -> System: generate new student number

 System -> System: fulfil <pUW + new student number>

 else else

 rnote over User, System #white: …..

 end

end

System -> User: result

@enduml

**Figure 2.8**

@startuml

start

:USER WISH;

:USER STORY;

repeat

repeat

:SCENARIO;

: tSSD ;

:INTEGRATION;

: NL-TEXT \n(and gSSD);

:VALIDATION;

repeat while (OK?) is ( NO\n)

->YES\n\n;

: IMPLEMENTATION

(SW PROCEDURES) ;

repeat while ( MORE\nSCENARIOS?) is ( YES\n)

->NO\n\n;

end

@enduml

# 4 Domain Modelling

## 4.1 Domain Models

**1st figure of Section 4.1**

@startuml

hide circle

hide members

class Concept

@enduml

**2nd figure of Section 4.1**

@startuml

hide circle

hide methods

class Concept{

Property 1

Property 2

⁝

⁝

}

@enduml

**3rd figure of Section 4.1**

@startuml

hide circle

hide members

" Concept A " "m " - " n" " Concept B ": xxx >

@enduml

**4th figure of Section 4.1**

Same as 3rd figure of Section 4.1

**5th figure of Section 4.1**

@startuml

hide circle

hide members

" Concept A " "m " - " n" " Concept B ": < is xxx-ed by

@enduml

**6th figure of Section 4.1**

@startuml

hide circle

hide methods

class Student{

Name

Address

Phone nr.

}

class Course{

Name

}

class Exam{

Date

}

Student " \* " - " \*" Course : enrols for >

Course "1 " - " \*" Exam : < belongs to

@enduml

**7th figure of Section 4.1**

@startuml

hide circle

hide members

" A " "m " - " n" " B ": xs >

@enduml

**8th figure of Section 4.1**

Same as 7th figure of Section 4.1

**9th figure of Section 4.1**

Same as 7th figure of Section 4.1

**10th figure of Section 4.1**

@startuml

hide circle

hide empty members

class Book{

ISBN

Title

Author

{field} Branch (the book is in)

{field} Book ID (used in that branch)

Publisher

Physical Condition

}

Member - Book : borrows >

Book -- Author: < wrote

Book -- Publisher: < published

Book - Branch: is in >

@enduml

## 4.2 Finding/Discovering/Determining Relevant Candidates

**1st figure of Section 4.2**

@startuml

hide circle

hide members

Student – Exam: enrols for >

Exam – Course: for >

Course – Lecturer: of >

@enduml

**2nd figure of Section 4.2**

@startuml

hide circle

hide members

Student – Exam: enrols for >

Exam – Course: for >

Course – Lecturer: of >

Student –- Grade: < of

Exam –- Grade: < for

@enduml

### 4.3.1 Many-to-Many Associations

**1st figure of Section 4.3.1**

@startuml

hide circle

hide members

" A " " \* " - " \*" " B ": xs >

@enduml

**2nd figure of Section 4.3.1**

@startuml

hide circle

hide members

" A " " 1 " - " \*" " C ": α <

" C " " \* " - " 1" " B ": β >

@enduml

**3rd figure of Section 4.3.1**

@startuml

hide circle

hide members

" Student " " 1 " - " \*" " Enrolment ": of <

" Enrolment " " \* " - " 1" " Course ": for >

@enduml

**4th figure of Section 4.3.1**

@startuml

hide circle

hide members

" Member " " 1 " - " \*" " Loan ": by <

" Loan " " \* " - " 1" " Book ": of >

@enduml

### 4.3.2 Ternary Associations and Beyond

**1st figure of Section 4.3.2**

@startuml

hide circle

hide members

Student " 1" -- "\*" " Rating ": by <

Lecturer " 1" -- " \*" " Rating ": of <

Course " 1" -- "\*" " Rating ": in <

@enduml

**2nd figure of Section 4.3.2**

@startuml

hide circle

hide members

class NV as “ Nominalized Verb ”

"Concept 0" " 1" -- "\*" NV: α0 <

"Concept 1" " 1" -- "\*" NV: α1 <

class "- - -" #white ##[dotted]white

"- - -" -[dotted]- NV

"Concept n" " 1" -- "\n \*" NV: αn <

@enduml

**3rd figure of Section 4.3.2**

@startuml

hide circle

hide empty members

class " Arrival " {

Arrival time

}

" Ship " " 1" -- "\*" " Arrival ": of <

@enduml

### 4.3.3 Individual Items Versus ‘Catalogue’ Items

**1st figure of Section 4.3.3**

@startuml

hide circle

hide empty members

class “ Book Copy “ {

Book ID

Branch

Condition

}

class “ Book Title “ {

ISBN

Title

Author

Publisher

}

Member – “ Book Copy “ : borrows >

“ Book Title “ -- Author: < wrote

“ Book Title “ -- Publisher: < published

 “ Book Copy “ “\*” -- “1” “ Book Title “: < describes

 “ Book Copy “- Branch: is in >

@enduml

**2nd figure of Section 4.3.3**

@startuml

hide circle

hide members

"Individual\n Item" " \* " - " 1" " Catalogue\n Item": < describes

@enduml

**3rd figure of Section 4.3.3**

@startuml

hide circle

hide members

"Individual\n Item" " \* " - " 1" " Catalogue\n Item": is described by >

@enduml

**4th figure of Section 4.3.3**

@startuml

hide circle

hide members

Client " \* " - " \*" Item: buys >

Item " \* " - " 1" Cashier: scans <

@enduml

### 4.3.4 Directed Graphs

**1st figure of Section 4.3.4**

@startuml

hide circle

hide members

Node “1” --- “\*” Arrow: < from

Node “1” --- “\*” Arrow: < to

@enduml

**2nd figure of Section 4.3.4**

@startuml

hide circle

hide empty members

class " Airport "{

Name

}

class Flight{

Airline

Duration

}

" Airport " “1” --- “\*” Flight: < from

" Airport " “1” --- “\*” Flight: < to

@enduml

**3rd figure of Section 4.3.4**

@startuml

hide circle

hide empty members

class Association{

Description

From-multiplicity

To-multiplicity

}

class " Concept "{

Concept name

}

class " Property "{

Property name

}

" Concept " “1” --- “\*” Association: < from

" Concept " “1” --- “\*” Association: < to

" Concept " “1” --- “\*” " Property ": < of

@enduml

### 4.3.5 Trees and the Like

**1st figure of Section 4.3.5**

@startuml

hide circle

hide members

class Root

Root <-- “Folder A"

Root <-- “Folder B"

Root <-- “Folder C"

Root <-- “File 1"

“Folder A" <-- “File 2"

“Folder A" <-- “File 3"

“Folder A" <-- “File 4"

“Folder C" <-- “Folder D"

“Folder C" <-- “File 5"

“Folder D" <-- “File 6"

“Folder D" <-- “File 7"

@enduml

**2nd figure of Section 4.3.5**

@startuml

hide circle

hide empty members

" Folder " “0..1” ---- “\*” " Folder ": ▲ is in

" Folder " “1” -- “\*” " File ": is in <

@enduml

**3rd figure of Section 4.3.5**

@startuml

hide circle

hide empty members

" Node " “0..1” - “\*” " Node ": ▲ points to

@enduml

### 4.3.6 Other Concepts Related to Themselves

**1st figure of Section 4.3.6**

@startuml

hide circle

hide empty members

" Person " “\*” ---- “\*” " Person ": ▲ once married with

@enduml

**2nd figure of Section 4.3.6**

@startuml

hide circle

hide empty members

" Product " “\*” ---- “\*” " Product ": ▲ is direct part of

@enduml

**3rd figure of Section 4.3.6**

@startuml

hide circle

hide empty members

class " Marriage "{

Wedding Day

}

" Person " "1" - "\*" " Marriage ": “ ◄ bride “

" Person " "1" - "\*" " Marriage ": “ ◄ groom “

@enduml

**4th figure of Section 4.3.6**

@startuml

hide circle

hide empty members

class " BOM-entry "{

Quantity Needed

}

" Product " "1" - " \* " " BOM-entry ": “ ◄ child “

" Product " "1" - "\*" " BOM-entry ": “ ◄ parent “

@enduml

**5th figure of Section 4.3.6**

@startuml

hide circle

hide empty members

" C " “\*” ---- “\*” " C "

@enduml

**6th figure of Section 4.3.6**

@startuml

hide circle

hide empty members

" C " "1" - "\*" " HC ": “ ◄ roleB “

" C " "1" - "\*" " HC ": “ ◄ roleA “

@enduml

### 4.3.7 Generalization and Specialization

**1st figure of Section 4.3.7**

@startuml

hide circle

hide empty members

Concept <|-- Subconcept

@enduml

**2nd figure of Section 4.3.7**

@startuml

hide circle

hide empty members

Concept <|-- Subconcept: “ is\_a”

@enduml

**3rd figure of Section 4.3.7**

@startuml

hide circle

hide empty members

Concept “1 ” <|-- “0..1” Subconcept

@enduml

**4th figure of Section 4.3.7**

@startuml

hide circle

hide empty members

class " Marriage "{

Wedding Day

}

" Person " <|-- " Woman "

" Person " <|-- " Man "

" Woman " "1" --- "\*" " Marriage ": bride <

" Man " "1" --- "\*" " Marriage ": groom <

@enduml

**5th figure of Section 4.3.7**

@startuml

hide circle

hide empty members

class " Marriage "{

Wedding Day

}

" Person " "1" - "\*" " Marriage ": “ ◄ partner2 “

" Person " "1" - "\*" " Marriage ": “ ◄ partner1 “

@enduml

## 4.4 Summary

**1st figure of Section 4.4**

@startuml

hide circle

hide members

" Concept A " - " Concept B ": xxx >

@enduml

**2nd figure of Section 4.4**

@startuml

hide circle

hide methods

class " Concept A " {

Property A1

Property A2

⁝

⁝

}

class " Concept B " {

Property B1

Property B2

⁝

⁝

}

" Concept A " "m " - " n" " Concept B ": xxx >

@enduml

# 5 Conceptual Data Models

## 5.2 From *Domain Models* to *Conceptual Data Models*: An Example

**1st figure of Section 5.2**

Same as 6th figure of Section 4.1

@startuml

hide circle

hide methods

class Student{

Name

Address

Phone nr.

}

class Course{

Name

}

class Exam{

Date

}

Student " \* " - " \*" Course : enrols for >

Course "1 " - " \*" Exam : < belongs to

@enduml

**2nd figure of Section 5.2**

@startuml

hide circle

hide methods

class Student{

Name

Address

Phone nr.

}

class Enrolment{

Date

}

class Course{

Name

}

class Exam{

Date

}

Student " 1 " - " \*" Enrolment : of <

Enrolment " \* " - " 1" Course : for >

Course "1 " - " \*" Exam : < belongs to

@enduml

**3rd figure of Section 5.2**

@startuml

hide circle

hide members

" A " " \* " - " 1" " B ": xs >

@enduml

**4th figure of Section 5.2**

@startuml

hide circle

hide empty members

class " A "{

^B

}

" A "}->" B ": xs >

@enduml

**5th figure of Section 5.2**

@startuml

hide circle

hide methods

class Student{

Name

Address

Phone nr.

}

class Enrolment{

^Student

^Course

Date

}

class Course{

Name

}

class Exam{

^Course

Date

}

Student <-{ Enrolment : of <

Enrolment }-> Course : for >

Course <-{ Exam : < belongs to

@enduml

**6th figure of Section 5.2**

@startuml

hide circle

hide methods

class Student{

! Number

Name

Address

[ Phone nr. ]

}

class Enrolment{

! ^Student

! ^Course

! Date

}

class Course{

! Code

% Name

% Faculty

}

class Exam{

! ^Course

! Date

}

Student <-{ Enrolment : of

Enrolment }-> Course : for

Course <-{ Exam : belongs to

@enduml

**7th figure of Section 5.2**

Same code as for 6th figure of Section 5.2 but with ‘hide methods’ replaced by ‘hide members’

## 5.3 From *Domain Models* to *Conceptual Data Models*: General Case

**1st figure of Section 5.3**

@startuml

hide circle

hide empty members

class " C " as C

class C #header:orange {

^A1

^A2

}

class " A1 " as A1

A1 <--{ C: <its A1>

class " A2 " as A2

A2 <--{ C: <its A2>

class " B1 " as B1 {

^C

}

C <--{ B1: <its C>

class " B2 " as B2 {

^C

}

C <--{ B2: <its C>

@enduml

**2nd figure of Section 5.3**

@startuml

hide circle

hide empty members

class " C " as C

class C #header:orange {

^A1

^A2

}

class " A1 " as A1

A1 <--{ C

class " A2 " as A2

A2 <--{ C

class " B1 " as B1 {

^C

}

C <--{ B1

class " B2 " as B2 {

^C

}

C <--{ B2

@enduml

**3rd figure of Section 5.3**

Same as 1st figure of Section 4.3.3

## 5.4 Using Arrows for Conceptual Data Models

**1st figure of Section 5.4**

@startuml

hide circle

hide members

" A " "m " - " n" " B ": xs >

@enduml

**2nd figure of Section 5.4 (m = ‘\*’ and n = ‘\*’)**

@startuml

hide circle

hide empty members

class " C "{

^A

^B

}

" A "<-{" C "

" C "}->" B "

@enduml

**3rd figure of Section 5. 4 (m = ‘\*’ and n = ‘1’)**

@startuml

hide circle

hide empty members

class " B "{

}

class " A "{

^B

}

" A "}->" B "

@enduml

**4th figure of Section 5. 4 (m = ‘\*’ and n = ‘0..1’)**

@startuml

hide circle

hide empty members

class " A "{

[ ^B ]

}

" A "}.>" B "

@enduml

**5th figure of Section 5. 4 (m = ‘1’ and n = ‘\*’)**

@startuml

hide circle

hide empty members

class " B "{

^A

}

" A "<-{" B "

@enduml

**6th figure of Section 5. 4 (m = ‘1’ and n = ‘1’)**

@startuml

hide circle

hide empty members

class " B "{

!^A

}

class " A "{

!^B

}

" A "->" B "

" A "<-" B "

@enduml

**7th figure of Section 5. 4 (m = ‘1’ and n = ‘1’)**

@startuml

hide circle

hide empty members

class " A&B "{

<A-properties>

<B-properties>

}

@enduml

**8th figure of Section 5. 4 (m = ‘1’ and n = ‘0..1’)**

@startuml

hide circle

hide empty members

class " B "{

!^A

}

class " A "{

}

" A "<-" B "

@enduml

**9th figure of Section 5. 4 (m = ‘0..1’ and n = ‘\*’)**

@startuml

hide circle

hide empty members

class " B "{

[ ^A ]

}

" A "<.{" B "

@enduml

**10th figure of Section 5. 4 (m = ‘0..1’ and n = ‘1’)**

@startuml

hide circle

hide empty members

class " B "{

}

class " A "{

!^B

}

" A "->" B "

@enduml

**11th figure of Section 5. 4 (m = ‘0..1’ and n = ‘0..1’)**

@startuml

hide circle

hide empty members

class " C "{

! ^A

% ^B

}

" A "<-" C "

" C "->" B "

@enduml

**12th figure of Section 5. 4 (also m = ‘0..1’ and n = ‘0..1’)**

@startuml

hide circle

hide empty members

class " B "{

}

class " A "{

[ !^B ]

}

" A ".>" B "

@enduml

**13th figure of Section 5. 4 ( also m = ‘0..1’ and n = ‘0..1’)**

@startuml

hide circle

hide empty members

class " B "{

[ !^A ]

}

class " A "{

}

" A "<." B "

@enduml

**14th figure of Section 5.4**

@startuml

hide circle

hide empty members

" C " <--{ " D "

" E " <..{ " F "

" G " <-- " H "

" K " <.. " L "

@enduml

**15th figure of Section 5.4**

@startuml

hide circle

hide empty members

Class Subconcept {

! ^Concept

}

Concept <|-- "Subconcept"

@enduml

### 5.6.1 Transforming Many-to-Many Associations

**1st figure of Section 5.6.1**

@startuml

hide circle

hide members

" A " " \* " - " \*" " B ": xs >

@enduml

**2nd figure of Section 5.6.1**

@startuml

hide circle

hide empty members

" A " <-{ " C ": α <

" C " }-> " B ": β >

Class " C " {

^A

^B

}

@enduml

### 5.6.2 Treating Associations in General

**1st figure of Section 5.6.2**

@startuml

hide circle

hide members

Student " 1" -- "\*" " Rating ": by <

Lecturer " 1" -- " \*" " Rating ": of <

Course " 1" -- "\*" " Rating ": in <

@enduml

**2nd figure of Section 5.6.2**

@startuml

hide circle

hide empty members

Class Rating as " Rating " {

^Student

^Lecturer

^Course

value

}

Student <--{ Rating: < by

Lecturer <--{ Rating: < of

Course <--{ Rating: < in

@enduml

**3rd figure of Section 5.6.2**

@startuml

hide circle

hide members

class NV as “ New concept ”

"Concept 1" " 1" -- "\*" NV: β1 <

"Concept 2" " 1" -- "\*" NV: β2 <

class "- - -" #white ##[dotted]white

"- - -" -[dotted]- NV

"Concept n" " 1" -- "\n \*" NV: βn <

@enduml

**4th figure of Section 5.6.2**

@startuml

hide circle

hide empty members

class NV as “ New concept ” {

^ Concept 1

^ Concept 2

:

^ Concept n

:

}

"Concept 1" <--{ NV: β1 <

"Concept 2" <--{ NV: β2 <

class "- - -" #white ##[dotted]white

"- - -" -[dotted]- NV

"Concept n" <--{ NV: βn <

@enduml

**5th figure of Section 5.6.2**

@startuml

hide circle

hide empty members

class " Arrival " {

^Ship

Arrival time

}

" Ship " <--{ " Arrival ": of <

@enduml

**6th figure of Section 5.6.2**

@startuml

hide circle

hide empty members

Class Rating as " Rating " {

^Student

^Class

value

}

Student <--{ Rating

Class Class as " Class " {

^Course

^Lecturer

}

Class <--{ Rating

Course <--{ Class

Lecturer <--{ Class

@enduml

### 5.6.3 Individual Items Versus ‘Catalogue’ Items

**1st figure of Section 5.6.3**

@startuml

hide circle

hide empty members

"Individual Item" }-> " Catalogue Item": is described by >

Class "Individual Item" {

^Catalogue Item

}

@enduml

### 5.6.4 Directed Graphs

**1st figure of Section 5.6.4**

@startuml

hide circle

hide empty members

" Node " <--{ Arrow: from

" Node " <--{ Arrow: to

class Arrow{

! ^Node from

! ^Node to

:

}

@enduml

**2nd figure of Section 5.6.4**

@startuml

hide circle

hide empty members

class A as “ A ”

class B as “ B ”

B <--{ A: r1

B <--{ A: r2

@enduml

**3rd figure of Section 5.6.4**

@startuml

hide circle

hide members

class A

A--> B: L1

A--> B: L2

A--> C: L1

C--> D: L1

C--> A: L4

C--> E: L3

B--> C: L1

D--> F: L5

E--> F: L5

@enduml

**4th figure of Section 5.6.4**

@startuml

hide circle

hide members

class Announced

Announced --> Confirmed

Announced --> Open

Announced --> Closed

Confirmed --> Open

Confirmed --> Closed

Open --> Suspended

Open --> Completed

Suspended --> Open

Completed --> Checked

Checked --> Closed

@enduml

**5th figure of Section 5.6.4**

@startuml

hide circle

hide empty members

class Node as “Allowed Status”

class Arrow as “Allowed Status Change”

Node <--{ Arrow: old \nstatus

Node <--{ Arrow: new \nstatus

Node <--{ Order: current \nstatus

class Node{

! Status

}

class Arrow{

! ^Status before

! ^Status after

}

class Order{

! Order ID

^Status

Remarks

}

@enduml

### 5.6.5 Trees and the Like

**1st figure of Section 5.6.5**

@startuml

hide circle

hide empty members

Node <..{ Node: points to

class Node{

 [ ^Node pointing to ]

}

@enduml

**2nd figure of Section 5.6.5**

@startuml

hide circle

hide empty members

class Node as “Employee”

Node <..{ Node: reports to

class Node{

! Emp-ID

[ ^Boss ]

}

@enduml

**3rd figure of Section 5.6.5**

@startuml

hide circle

hide members

class E0

E1 --> E0

E2 --> E0

E3 --> E1

E4 --> E1

E5 --> E1

E6 --> E2

E7 --> E2

@enduml

**4th figure of Section 5.6.5**

@startuml

hide circle

hide empty members

class Node as “Employee”

Node <..{ Node: reports to

Node <..{ Node: works for

class Node{

! Emp-ID

[ ^Functional Boss ]

[ ^Project Boss ]

}

@enduml

### 5.6.6 Other Concepts Related to Themselves

**1st figure of Section 5.6.6**

@startuml

hide circle

hide empty members

" Concept " “\*” ---- “\*” " Concept ": ▲ xxs

@enduml

**2nd figure of Section 5.6.6**

@startuml

hide circle

hide empty members

Class xx-ment {

^Concept xx-ing

^Concept xx-ed

}

" Concept " "1" - "\*" “xx-ment”: " ◄ xx-ing "

" Concept " "1" - "\*" “xx-ment”: " ◄ xx-ed "

@enduml

**3rd figure of Section 5.6.6**

@startuml

hide circle

hide empty members

Class xx-ment {

^Concept xx-ing

^Concept xx-ed

}

" Concept " <-{ “xx-ment”: " ◄ xx-ing "

" Concept " <-{ “xx-ment”: " ◄ xx-ed "

@enduml

**4th figure of Section 5.6.6**

@startuml

hide circle

hide empty members

" Product " “\*” ---- “\*” " Product ": ▲ contains

@enduml

**5th figure of Section 5.6.6**

@startuml

hide circle

hide empty members

Class Containment {

^Product containing

^Product contained

}

" Product " <-{ “Containment”: " ◄ containing "

" Product " <-{ “Containment”: " ◄ contained "

@enduml

**6th figure of Section 5.6.6**



Note that we can enter a table in Plantuml:

@startuml

hide circle

hide members

class Legend: #line.bold

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parent** | <-- | **Child** | **:** | **#** |
| Storage\_wall | <-- | Normal\_drawer | : | 3 |
| Storage\_wall | <-- | Chest | : | 1 |
| Storage\_wall | <-- | Supports | : | 16 |
| Storage\_wall | <-- | Cupboard | : | 2 |
| Storage\_wall | <-- | Shelf | : | 8 |
| Chest | <-- | Normal\_drawer | : | 4 |
| Chest | <-- | Thumb\_screw | : | 16 |
| Chest | <-- | Supports | : | 8 |
| Chest | <-- | Surround | : | 1 |
| Cupboard | <-- | Supports | : | 8 |
| Cupboard | <-- | Surround | : | 1 |
| Cupboard | <-- | Thumb\_screw | : | 16 |
| Cupboard | <-- | Shelf | : | 4 |
| Normal\_drawer | <-- | Bolt\_and\_nut | : | 6 |
| Normal\_drawer | <-- | Drawer\_board | : | 7 |
| Normal\_drawer | <-- | Thumb\_screw | : | 2 |
| Supports | <-- | Thumb\_screw | : | 3 |
| Supports | <-- | Support\_part | : | 2 |

@enduml

### 5.6.7 Generalization and Specialization

**1st figure of Section 5.6.7**

@startuml

hide circle

hide empty members

Class Subconcept {

! ^Concept

}

Concept <|-- "Subconcept"

@enduml

# 6 Directions for Implementation

## 6.3 Interaction with a (Software) System

**Figure 6.3**

@startuml

User -> System: A

System -> System: B

System -> User: C

@enduml

**Figure 6.4**

@startuml

User -> Interface: A

Interface -> “ Kernel “: A´

“ Kernel “ -> “ Kernel “: B´

“ Kernel “ -> Interface: C´

Interface -> User: C

box "System"

participant Interface

participant “ Kernel “

end box

@enduml

**Figure 6.5**

@startuml

participant U as “ User ”

box "System"

participant I as “Interface”

participant K as “ Kernel “

end box

U -> I : A

I -> I : B

I -> U : C

@enduml

### 6.4.1 From *Conceptual Data Models* to *Data Models in SQL*

**Figure 6.6 (a)**

@startuml

hide circle

hide empty members

class SE as “Study Enrolment” {

! Student

! Study

}

class Course {

! Course code

% Course name

% Study

}

class CE as “Course Enrolment” {

! ^ Study Enrolment

! ^ Course

}

class Exam {

! ^ Course

! Exam date

}

class EE as “Exam Enrolment” {

! ^ Course Enrolment

! ^ Exam

}

Exam <--{ EE: for

CE <--{ EE: within

Course <--{ Exam: for

Course <--{CE: in

SE <--{ CE: within

@enduml

**Figure 6.6 (b)**

@startuml

hide circle

hide empty members

class SE as “Study Enrolment” {

! Student

! Study

}

class Course {

! Course code

% Course name

% Study

}

class CE as “Course Enrolment” {

! ^ Student, Study

! ^ Course

}

class Exam {

! ^ Course

! Exam date

}

class EE as “Exam Enrolment” {

! ^ Student, Study, Course

! ^ Course, Exam date

}

Exam <--{ EE: for

CE <--{ EE: within

Course <--{ Exam: for

Course <--{CE: in

SE <--{ CE: within

@enduml

## 6.5 Generating an Initial Class Diagram

**Figure 6.7**

@startuml

Class A1 as "Concept A"

Class A2 as "Concept B" {

^ Concept A

}

Class A3 as "Concept C" {

[ ^ Concept B ]

}

Class A4 as "Concept D" {

! ^ Concept E

[ % ^ Concept C ]

}

Class A5 as "Concept E"

Class A6 as "Concept F" {

! ^ Concept E

}

A1 <--{ A2 : many-to-1 <

A2 <.{ A3 : many-to-0/1 <

A4 ..> A3 : 0/1-to-0/1 >

A4 -> A5 : > \n 0/1-to-1

A5 <|-- A6 : sub-concept \n arrow

@enduml

**Figure 6.8**

@startuml

Class A5 as "Concept E" {

! E-id

PropA

/ PropB

MethA()

MethB(x)

MethC(y,z)

}

Class A6 as "Concept F" {

! ^ Concept E

MethD(u,v,w)

PropC

}

A5 <|-- A6 : sub-concept \n arrow

@enduml

# 7 Organizing and Managing the Development Process

### 7.2.1 Waterfall Methods

**Figure 7.2**

@startuml

|Phase 1|

: Analysis ;

|Phase 2|

: Design ;

|Phase 3|

:Implementation;

|Phase 4|

: Verification ;

|Phase 5|

: Maintenance ;

@enduml

### 7.2.2 Parallel Development

**Figure 7.3**

@startuml

#99FF99: Analysis ;

: Design ;

:Implementation;

: Verification ;

: Maintenance ;

@enduml

**Figure 7.4**

@startuml

#99FF99: Analysis ;

: Global Design ;

fork

: Design

 Subproject 1 ;

:Implementation

 Subproject 1 ;

fork again

: Design

 Subproject 2 ;

:Implementation

 Subproject 2 ;

fork again

:. . . . . . .

 ;

:. . . . . . .

 ;

fork again

: Design

 Subproject n ;

:Implementation

 Subproject n ;

endfork

#aqua: Integration ;

: Verification ;

: Maintenance ;

@enduml

### 7.2.4 Prototyping

**Figure 7.5**

@startuml

start

repeat

#99FF99: Analysis ;

: Design ;

: Prototype

Implementation;

#99FF99: Prototype

 Evaluation ;

repeat while (users satisfied?) is ( no\n)

->yes\n\n;

: Implementation on

operational platform;

: Maintenance ;

@enduml

**Figure 7.6**

@startuml

start

repeat

#99FF99: Analysis ;

: Design ;

: Prototype

Implementation;

#99FF99: Prototype

 Evaluation ;

repeat while (users satisfied?) is ( no\n)

->yes;

#aqua: Redesign for

operational platform;

: Implementation on

operational platform;

: Maintenance ;

@enduml

### 7.2.5 Iterative Development and Incremental Development

**Figure 7.7**

Three times the following Plantuml-drawing, within a table:

@startuml

#99FF99: Analysis ;

: Design ;

:Implementation;

: Verification ;

: Maintenance ;

@enduml

**Figure 7.8**

@startuml

start

repeat

#99FF99: Analysis ;

: Design ;

: Prototype Implementation ;

#99FF99: Prototype Evaluation ;

repeat while ( satisfying? ) is (no)

->yes;

: Implementation on operational platform ;

: Maintenance ;

@enduml

**Figure 7.9**

Three times the following Plantuml-drawing:

@startuml

start

repeat

#99FF99: Analysis ;

: Design ;

: Prototype

Implementation;

#99FF99: Prototype

 Evaluation ;

repeat while (satisfying?) is (no)

->yes;

: Implementation on

operational platform;

: Maintenance ;

@enduml

# 8 A Non-Trivial University Example Worked out

## 8.3 A Simple Domain Model for the Situation

**Figure 8.1**

@startuml

hide circle

hide empty members

Student "\*" -- "\*" Study: enrols for >

Student "\*" -- "\*" Course: enrols for >

Student "\*" -- "\*" Course: participates in >

Student "\*" -- "\*" Exam: enrols for >

Student "\*" -- "\*" Exam: gets graded for >

Student "\*" -- "\*" Course: rates >

Study "1" -- "\*" Course: within <

Course "1" -- "\*" Exam: < for

Course "\*" -- "\*" Course: requires >

@enduml

## 8.4 From Domain Model to an Initial Conceptual Data Model

**Figure 8.2**

@startuml

hide circle

hide empty members

class SE as “Study Enrolment”

class CE as “Course Enrolment”

class EE as “Exam Enrolment”

Study <--{ Course: within

Course <--{ Exam: for

Student <--{ SE: of

SE }--> Study: for

Student <--{ CE: of

CE }--> Course: in

Student <--{ Participation: of

Participation }--> Course: in

Student <--{ EE: of

EE }--> Exam: for

Student <--{ Grading: of

Grading }--> Exam: for

Student <--{ Rating: by

Rating }--> Course: of

Course <--{ Requirement: required \ncourse

Course <--{ Requirement: for

@enduml

## 8.5 A Refined Conceptual Data Model Including References

**Figure 8.3**

@startuml

hide circle

hide members

class SE as "Study Enrolment"

class CE as "Course Enrolment"

class EE as "Exam Enrolment"

class GR as "Grading"

class PA as "Participation"

class RA as "Rating"

RA -> PA: 7b

PA -> CE: 3b

SE <-- CE: 2c

CE <- EE: 5b

EE <- GR: 6b

@enduml

**Figure 8.4**

@startuml

hide circle

hide empty members

class SE as “Study Enrolment” {

^ Student

^ Study

}

class Course {

^ Study

}

class CE as “Course Enrolment” {

^ Study Enrolment

^ Course

}

class Exam {

^ Course

}

class Participation {

^ Course Enrolment

}

class Rating {

^ Participation

}

class EE as “Exam Enrolment” {

^ Course Enrolment

^ Exam

}

class Grading {

^ Exam Enrolment

}

class Requirement {

^ Requiring course

^ Required course

}

Exam <--{ EE: for

CE <--{ EE: within

EE <--{ Grading: provided

Study <--{ Course: within

Course <--{ Exam: for

Student <--{ SE: of

Study <--{ SE: for

Course <--{CE: in

SE <--{ CE: within

Course <--{ Requirement: required \ncourse

Course <--{ Requirement: for

Participation }-> CE: after

Participation <--{ Rating: after

@enduml

## 8.7 Conceptual Data Model with ‘all’ Relevant Properties

**Figure 8.5**

@startuml

hide circle

hide empty members

class Faculty {

! Abbreviation

% Full name

Physical address

E-mail address

Phone number

Founding year

}

class Student {

! Student number

Name

Address

[ Gender ]

Birth date

[ Phone number ]

U-mail

Registration date

Left?

}

class Study {

! Study code

Level

ECTS-size

^ Faculty

}

class SE as “Study Enrolment” {

! ^ Student

! ^ Study

Enrolment date

}

class Course {

! Course code

% Course name

% ^ Study

Study year

Study block

ECTS-size

Description

^ Faculty

}

class CE as “Course Enrolment” {

! ^ Study Enrolment

! ^ Course

Enrolment date

}

class Exam {

! ^ Course

! Exam date

}

class Participation {

! ^ Course Enrolment

Nr. of Presences

}

class Rating {

! ^ Participation

Value given

}

class EE as “Exam Enrolment” {

! ^ Course Enrolment

! ^ Exam

Enrolment date

}

class Grading {

! ^ Exam Enrolment

Grade

}

class Requirement {

! ^ Requiring course

! ^ Required course

[ Explanation ]

}

Exam <--{ EE: for

CE <--{ EE: within

EE <-- Grading: provided

Study <--{ Course: within

Course <--{ Exam: for

Student <--{ SE: of

Study <--{ SE: for

Course <--{CE: in

SE <--{ CE: within

Course <--{ Requirement: required \ncourse

Course <--{ Requirement: for

Participation -> CE: after

Participation <-- Rating: after

Faculty <--{ Study: at

Faculty <--{ Course: responsible \nfaculty

@enduml

### 8.9.1 Informally

**Figure 8.6**

We added the yellow text, in particular ‘class X’ on the right of SE (Study Enrolment), in order to get our circle in the middle of the diagram. We made the background colour and line colour of ‘class X’ white, as well as the new arrow.

@startuml

hide circle

hide empty members

class SE as “Study Enrolment”

class CE as “Course Enrolment”

Study <--{ Course: within

Study <--{ SE: for

Course <--{CE: in

SE <--{ CE: within

class X as “<size:36>O " #white ##white

SE <-[#white] X

@enduml

**Figure 8.7**

We added the yellow text, in particular ‘class X’ on the right of Study, in order to get our **crossed-out** circle in the middle of the diagram. We made the background colour and line colour of ‘class X’ white, as well as the new arrow.

@startuml

hide circle

hide empty members

Faculty <-- Course: responsible \nfaculty

Study <--{ Course: within

Faculty <-- Study: at

class X as “<size:36>Ꝋ" #white ##white

Study <-[#white] X

@enduml

**Figure 8.8**

@startuml

hide circle

hide empty members

class A as “ A ”

class B as “ B ”

B <--{ A: r1\n<size:36>O

B <--{ A: r2\n\n\n

@enduml

## 8.11 Specification in SQL

**Figure 8.9**

@startuml

hide circle

hide empty members

class Faculty {

! Abbreviation

% Full name

Physical address

E-mail address

Phone number

Founding year

}

class Student {

! Student number

Name

Address

[ Gender ]

Birth date

[ Phone number ]

U-mail

Registration date

Left?

}

class Study {

! Study code

Level

ECTS-size

^ Faculty

}

class SE as “Study Enrolment” {

! ^ Student

! ^ Study

Enrolment date

}

class Course {

! Course code

% Course name

% ^ Study

Study year

Study block

ECTS-size

Description

^ Faculty

}

class CE as “Course Enrolment” {

! ^ Student, Study

! ^ Course

Enrolment date

}

class Exam {

! ^ Course

! Exam date

}

class Participation {

! ^ Student, Study, Course

Nr. of Presences

}

class Rating {

! ^ Student, Study, Course

Value given

}

class EE as “Exam Enrolment” {

! ^ Student, Study, Course

! ^ Course, Exam date

Enrolment date

}

class Grading {

! ^ Student, Study, Course

! ^ Course, Exam date

Grade

}

class Requirement {

! ^ Requiring course

! ^ Required course

[ Explanation ]

}

Exam <--{ EE: for

CE <--{ EE: within

EE <-- Grading: provided

Study <--{ Course: within

Course <--{ Exam: for

Student <--{ SE: of

Study <--{ SE: for

Course <--{CE: in

SE <--{ CE: within

Course <--{ Requirement: required \ncourse

Course <--{ Requirement: for

Participation -> CE: after

Participation <-- Rating: after

Faculty <--{ Study: at

Faculty <--{ Course: responsible \nfaculty

@enduml

# 9 Converting a Large Use Case

**Figure 9.1**

@startuml

hide circle

hide members

skinparam classBackgroundColor White

class tSSD as " textual \nSSD "

"Use Case\n<i> Process Sale</i> " -> tSSD: §9.1

tSSD -> " graphical \nSSD ": §9.3

tSSD --> " Natural \n Language " : §9.2

@enduml

## 9.3 Converting a Large Textual SSD to a Graphical SSD

**ProcessSale**

@startuml

title ProcessSale \n

Customer -> Customer: arrive at checkout \nwith items to purchase

Cashier -> System: StartNewSale

System -> System: CreateSale

group repeat

 group perform

 rnote over Cashier, System #white: EnterItem

 end

group in any order

System -> System: RecordSaleLineItem

 System -> Cashier: description, price, \nand running total

end

rnote over Cashier #white: \*\*until\*\* cashier indicates done

end

Cashier -> System: EndSale

System -> Cashier: total with taxes

Cashier -> Customer: total

Cashier -> Customer: request for payment

group maybe

group perform

rnote over Cashier, System #white: [HandleDiscount](#handleDiscount2)

end

end

group in any order

group maybe

group perform

rnote over Cashier, System #white: [HandleCredit](#handleCredit2)

end

end

group maybe

group perform

rnote over Cashier, System #white: [HandleCoupons](#handleCoupons2)

end

end

end

group perform

rnote over Customer, AutSys #white: [HandlePayment](#handlePayment2)

end

group in any order

System -> System: log completed sale

System -> AccSys: sale and payment infoc

System -> InvSys: sale and payment info

end

group maybe

group perform

rnote over Cashier, System #white: [HandleGiftReceipt](#handleGiftReceipt2)

end

end

group if [ system detects printer is out of paper ]

group perform

rnote over Cashier, System #white: [HandlePaperShortage](#handlePaperShortage2)

end

 end

System -> Cashier: receipt

Customer -> Customer: leave with receipt and goods

@enduml

**EnterItem**

@startuml

title EnterItem \n

group either

group perform

rnote over Cashier, System #white: [HandleCodedItem](#handleCodedItem)(item-ID [; q])

end

else or

Cashier -> System: enterPricedItem(P-category▼; price)

else or

Cashier -> System: enterWeightItem(W-category▼; weight)

end

@enduml

**HandleCodedItem**

@startuml

participant Cas as “Cashier”

participant Sys as “System”

title HandleCodedItem(item-ID [; q]) \n

Cas -> Sys: enterCodedItem(item-ID [; q])

group if Sys: quantity parameter q is absent

 Sys -> Sys: make q equal to 1

end

Sys -> Sys: determine item i having that item-ID

group if [ Sys: item-ID is unknown ]

 Sys -> Cas: “Unknown item ID”

 group if [ Cas: there is a human-readable item-ID ]

 Cas -> Sys: enterManually(item-ID; q)

 else else

group if [ Cas: there is a price on the tag ]

 Cas -> Sys: enterPrice(price; q)

 Cas -> Sys: applyStandardTaxation

 else else /\* after finally finding out the correct item-ID or price \*/

 group either

Cas -> Sys: enterManually(item-ID; q)

 else or

Cas -> Sys: enterPrice(price; q)

 Cas -> Sys: applyStandardTaxation

 end

 end

 end

end

@enduml

**HandleDiscount**

@startuml

title HandleDiscount \n

Cashier -> System: applyDiscount(Customer ID)

System -> System: apply discount to sale

System -> Cashier: new total with taxes

@enduml

**HandleCredit**

@startuml

title HandleCredit \n

Cashier -> System: applyCredit(Customer ID)

System -> System: apply credit to sale up to price = 0

System -> System: reduce remaining credit

System -> Cashier: new total with taxes

@enduml

**HandleCoupons**

@startuml

title HandleCoupons \n

group repeat

 Cashier -> System: record coupon

 System -> System: reduce price with value of coupon

 System -> System: record usage of coupon

 System -> Cashier: new total with taxes

rnote over Cashier #white: \*\*until\*\* coupons are done

end

@enduml

**HandlePayment**

@startuml

title HandlePayment \n

participant Customer

participant Cashier

participant System

group either

group perform

rnote over Customer, System #white: [HandleCashPayment](#handleCashPayment2)

end

else or

group perform

rnote over Customer, AutSys #white: [HandleCreditPayment](#handleCreditPayment2)

end

else or

group perform

rnote over Customer, System #white: [HandleCheckPayment](#handleCheckPayment2)

end

else or

group perform

rnote over Customer, System #white: [HandleDebitPayment](#HandleDebitPayment2)

end

end

@enduml

**HandleCashPayment**

@startuml

title HandleCashPayment \n

Cashier -> System: Enter(cash amount tendered)

System -> Cashier: balance due

System -> System: release cash drawer

Cashier -> System: deposit cash amount tendered

Cashier -> Customer: cash balance

System -> System: record cash payment

@enduml

**HandleCreditPayment**

@startuml

title HandleCreditPayment \n

Customer -> System: MakeCreditPay(credit account info)

System -> Cashier: payment info (for verification)

Cashier -> System: confirm

System -> AutSys: payment approved?

AutSys -> System: payment approval

group in any order

System -> Cashier: payment approved

System -> System: record credit payment

end

rnote over System, Cashier #white: \*\*. . .\*\* \n\*\*. . .\*\* \n\*\*. . .\*\*

@enduml

**HandleGiftReceipt**

@startuml

title HandleGiftReceipt \n

Cashier -> System: giveGiftReceipt

System -> Cashier: gift receipt

@enduml

**HandlePaperShortage**

@startuml

title HandlePaperShortage \n

System -> Cashier: “Out of paper”

Cashier -> Cashier: replace paper

Cashier -> System: printReceipt

@enduml

**HandleManagerOverride**

@startuml

title HandleManagerOverride \n

Manager -> System: changeModeTo(‘Manager’)

System -> System: change to mode ‘Manager’

Manager -> System: <do some ManagerMode operation>

System -> System: change to mode ‘Cashier’

@enduml

**HandleTaxExempt**

@startuml

title HandleTaxExempt \n

Cashier -> System: enterStatusCode(‘tax-exempt’)

System -> System: record Status Code ‘tax-exempt’

@enduml

**HandleItemRemoval**

@startuml

title HandleItemRemoval \n

Customer -> Cashier: request to remove item

group if [ value of item ≤ cashier-limit ]

 Cashier -> System: remove item

 else else

 Manager -> System: remove item

end

System -> System: delete item

System -> Cashier: new (running) total

@enduml

**HandleSaleCancellation**

@startuml

title HandleSaleCancellation \n

Cashier -> System: cancel sale

System -> System: delete sale

System -> Cashier: “Done”

@enduml

**HandleSaleSuspension**

@startuml

title HandleSaleSuspension \n

Cashier -> System: suspendSale

System -> System: record suspended sale

System -> Cashier: suspend receipt

@enduml

# 10 Development Example where Requirements Constantly Change

## 10.4 Variable Thresholds per Room Type

**Figure 10.1**

@startuml

hide circle

hide methods

package “Overview of configuration structure” {

“Room Type” <--{ Room

Room <--{ Sensor

Room <--{ Heating

Room <--{ Airco

}

class “Room Type” {

! RTID

Hmin

Hmax

Amin

Amax

}

class Room {

! RID

^ Room Type

}

class Sensor {

! SID

^ Room

}

class Heating {

! HID

^ Room

State

}

class Airco{

! AID

^ Room

State

}

@enduml

## 10.6 A Corresponding Textual SSD and Graphical SSD

**Figure 10.2**

@startuml

“sensor x” -> System: Measurement(x, t, y)

group in any order

System -> “External System”: Store(x, t, y, r)

participant h as “heating h”

participant a as “airco a”

group if [ t < Hmin of the type of room where sensor x is in ]

group for each [ heating h in the room of x in state ‘Off’ ]

System -> h: ‘On!’

System -> System: Change state of h to ‘On’

end

end

group if [ t > Hmax of the type of room where sensor x is in ]

group for each [ heating h in the room of x in state ‘On’ ]

System -> h: ‘Off!’

System -> System: Change state of h to ‘Off’

end

end

group if [ t < Amin of the type of room where sensor x is in ]

group for each [ airco a in the room of x in state ‘On’ ]

System -> a: ‘Off!’

System -> System: Change state of a to ‘Off’

end

end

group if [ t > Amax of the type of room where sensor x is in ]

group for each [ airco a in the room of x in state ‘Off’ ]

System -> a: ‘On!’

System -> System: Change state of a to ‘On’

end

end

end

@enduml

## **10.11 Interactions Between Our System and its Environment**

**Figure 10.3**

@startuml

participant Thresholder

participant “sensor x”

participant “\*\*System\*\*” #red

“\*\*System\*\*” -> “sensor x”**:** SendTemperature

“sensor x” -> “\*\*System\*\*”**:** Measurement(x, t, *y*)

“\*\*System\*\*” -> “Data Store”**:** Store(x, t, y, r)

“\*\*System\*\*” -> “heating h”**:** ‘On!’ / ‘Off!’

“heating h” -> “\*\*System\*\*”**:** Status(h)

“\*\*System\*\*” -> “airco a”**:** ‘On!’ / ‘Off!’

“airco a” -> “\*\*System\*\*”**:** Status(a)

Thresholder -> “\*\*System\*\*”**:** UpdateThresholds(r, *h1, h2, a1, a2*)

@enduml

## **10.12** **Looking Back: Typical Ingredients of an IS and a** Control System

**Figure 10.4**

@startuml

Actor -> System: CRUD-request

System -> System: CRUD-operation

System -> Actor: Result

…

@enduml

**Figure 10.5**

@startuml

“Sensor x” -> System: Measurement

System -> “Actuator y”: Command

…

“Actuator y” -> System: Feedback

@enduml

# Appendix: Our Plantuml Tutorial

All Plantuml-code can be found in the Appendix itself

## A.1 From Textual SSDs to Graphical SSDs

Contains two Plantuml-drawings

## A.2 Domain Models

Contains three Plantuml-drawings

## A.3 Conceptual Data Models

Contains four Plantuml-drawings

## A.4 Class Diagrams

Contains one Plantuml-drawing

## A.5 System as Interface + Kernel

Contains two Plantuml-drawings

## A.6 Use Case Diagrams

Contains one Plantuml-drawing