System Domain Analysis and Application Models

Introduction to Systems Engineering

I2ISE

Introduction

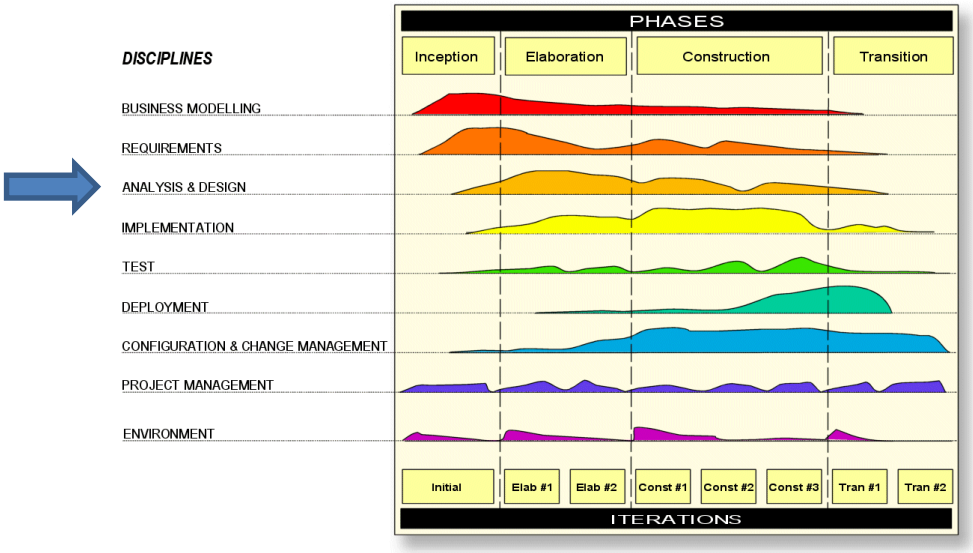
- What is *System Domain Analysis*?
- What is a *Domain Model?*
- Why create the Domain Model?
- How to create a Domain Model?

What is System Domain Analysis?

- *System Domain Analysis* (SDA) is an activity to analyse the system domain in order to find domain-specific concepts

- SDA is conducted between requirements and implementation (design)

- Prime artefact of SDA: The *Domain Model*



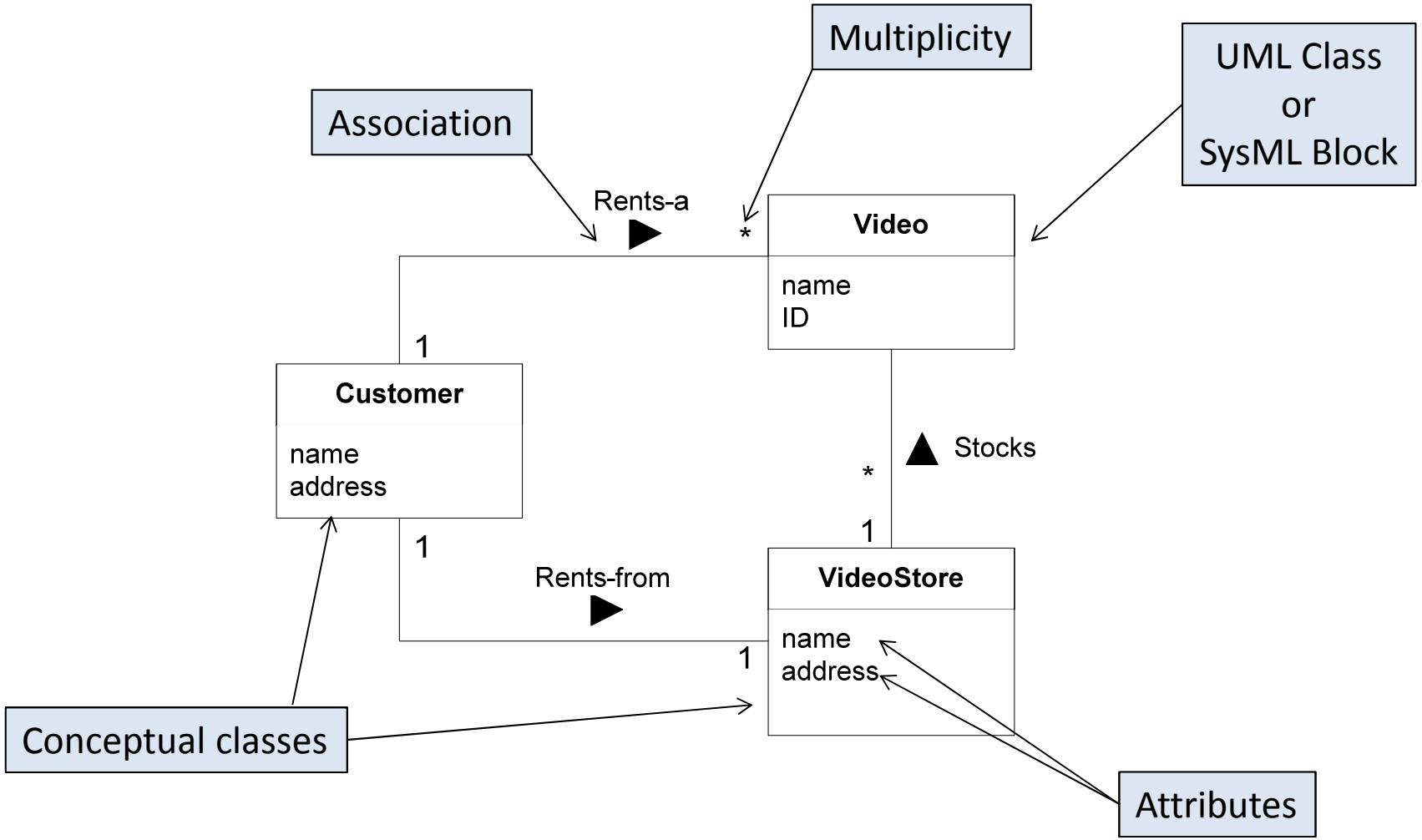
What is a Domain Model (and what is it not)?

- The Domain Model is an illustration of ”noteworthy concepts” in the system domain.
  - The concepts, their associations, multiplicity and attributes
  - Not responsibilities and operations!

- The Domain Model shows real-world concepts, *not* SW or HW entities
  - ”**`Bus`**”, ”**`Payment`**”, ”**`ATM`**” — **OK** – noteworthy concepts
  - ”**`SalesDatabase`**”, ”**`string`**” — **FAIL** – *software* entities

- The DM is a visual dictionary drawn as a UML class diagram
  - Everybody agree upon the names in the model
  - A ”new guy” can quickly pick up on terminology

Domain Model: Example



What is a conceptual class

- A conceptual class as shown on a Domain Model is an *idea*, a *thing* or an *object*

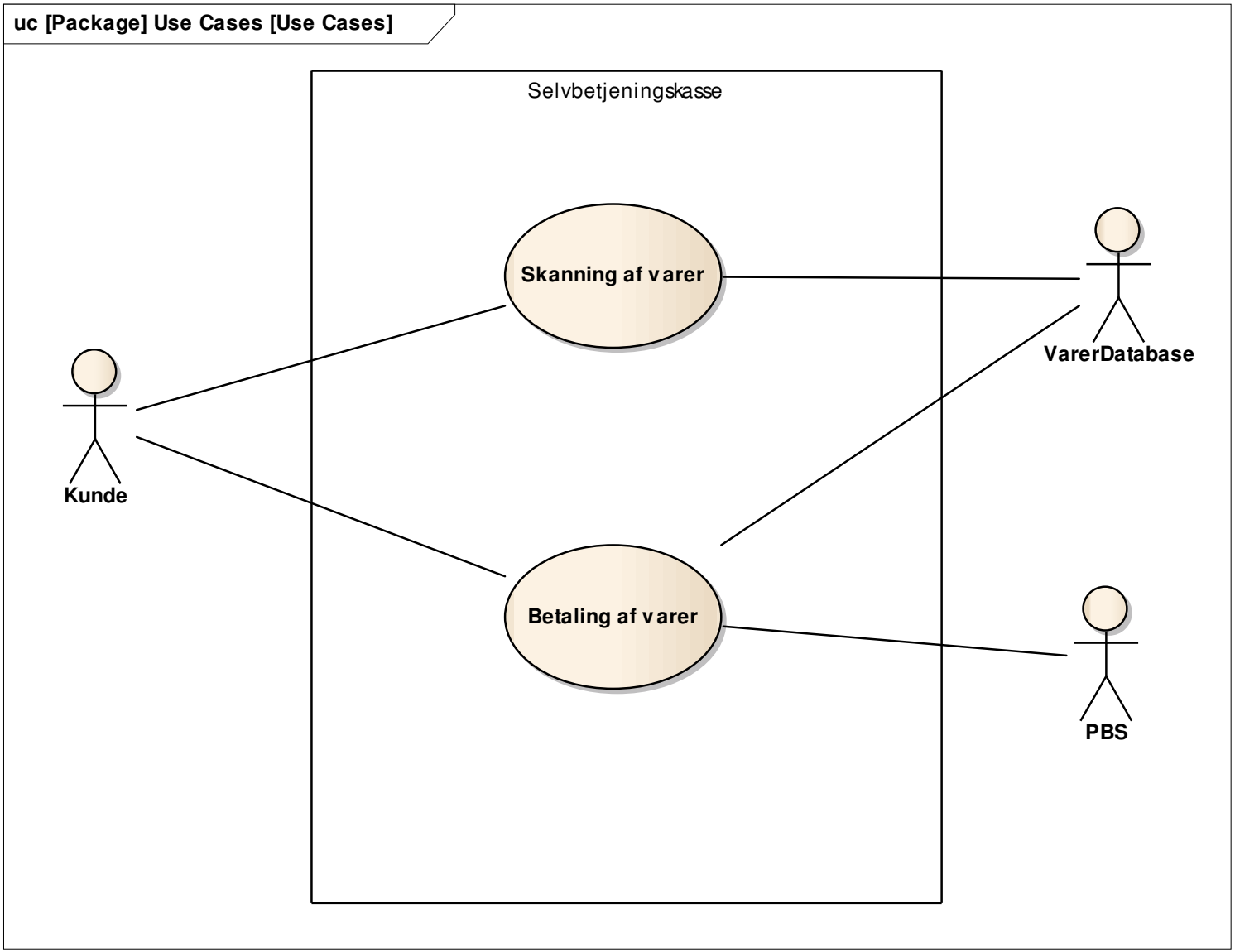
- Can also be defined by the class' *symbol*, *intension* and *extension*
  - Symbol: The word(s) or images representing the conceptual class
  - Intension: The definition of the conceptual class
  - Extension: The set of examples to which the conceptual class applies

Why create a Domain Model?

- Doing SDA and creating the Domain Model helps to identify key concepts and things to investigate
- The Domain Model aids the very hard step from requirements to design
  - The first step from ”what” to ”how”
- The Domain Model lowers the ”representational gap” between domain and implementation

Selvbetjeningskasse

Selvbetjeningskasse – Use Cases



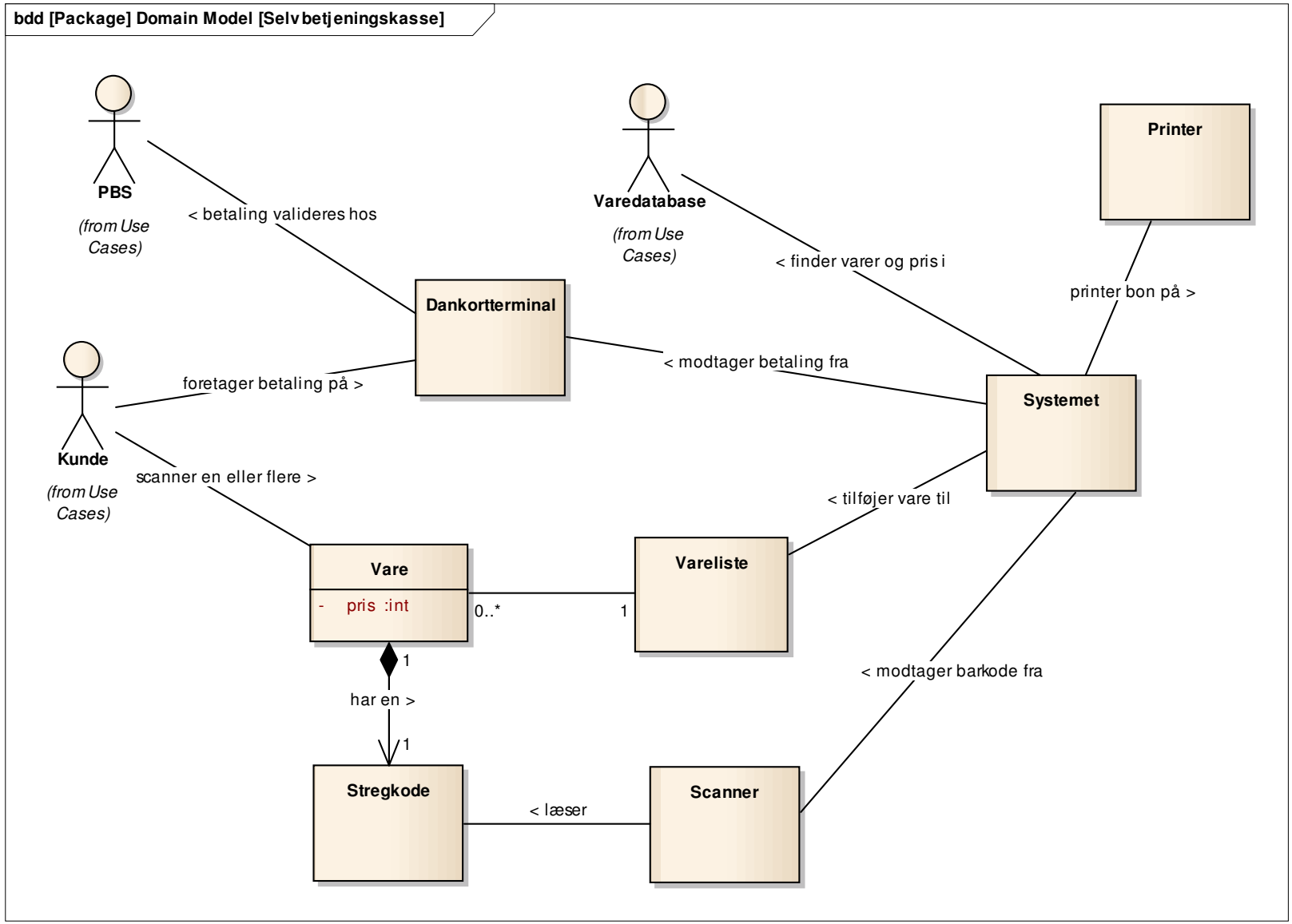
Scanning af Vare (Hovedscenarie)

1. Selvbetjeningskassen anmoder kunden om at skanne vare
2. Kunden placerer vare foran skanner
3. Systemet skanner varens stregkode
4. Systemet finder varens pris i varedatabasen
5. Vare med pris tilføjes til en vareliste
6. Kunden lægger vare i pose på bordet ved siden af skanner
7. Punkterne 1-6 gentages indtil alle varer er skannet
8. Kunden vælger afslut

Betaling af Vare (Hovedscenarie)

1. Kunden vælger betal med dankort på beløbet
2. Kunden indsætter kort i dankortterminalen
3. System viser det totale beløb og anmoder om pinkode
4. Kunden indtaster pinkode
5. Kort og pinkode valideres mod PBS
6. Printer udskriver bon med vareliste

Domain model



How to create Domain Models

- Creating a Domain Model is easy – creating a *useful* one is hard!

- Three steps to follow:

Step 1: Find the *conceptual classes* in the domain

Step 2: Draw the classes in a UML class diagram (or SysML Block Diagram)

Step 3: Identify associations and attributes between conceptual classes

Finding conceptual classes: Nouns

- From a textual description of requirements, one may also scan the text for *nouns* or *noun phrases* to find candidates
  - Again, not all nouns are good conceptual classes

- Exercise: From the below UC description…
  - Identify meaningful conceptual classes
  - Indetify nouns that are *not* meaningful conceptual classes

UC Rent Video: Main success scenario

1. Customer arrives at checkout counter with video
2. Cashier starts a new rental
3. Cashier scans member card's magnetic strip
4. Cashier scans video's bar code
5. System registers rental of video to Customer in ledger
6. Cashier requests due amount from Customer
7. Customer pays due amount
8. Cashier hands video to Customer

How to create Domain Models

- Creating a domain model is easy – creating a *useful* one is hard!

- Three steps to follow:

Step 1: Find the *conceptual classes* in the domain

Step 2: Draw the classes in a UML class diagram (or SysML Block Diagram)

Step 3: Identify associations and attributes between conceptual classes

Creating a Domain Model, step 2: Draw the conceptual classes

- The conceptual classes identified in Step 1 can now be drawn in a UML class diagram or SysML Block Diagram.

- CASE tool or whiteboard, your choice…

- But this diagram does not really become useful before Step 3

How to create Domain Models

- Creating a domain model is easy – creating a *useful* one is hard!

- Three steps to follow:

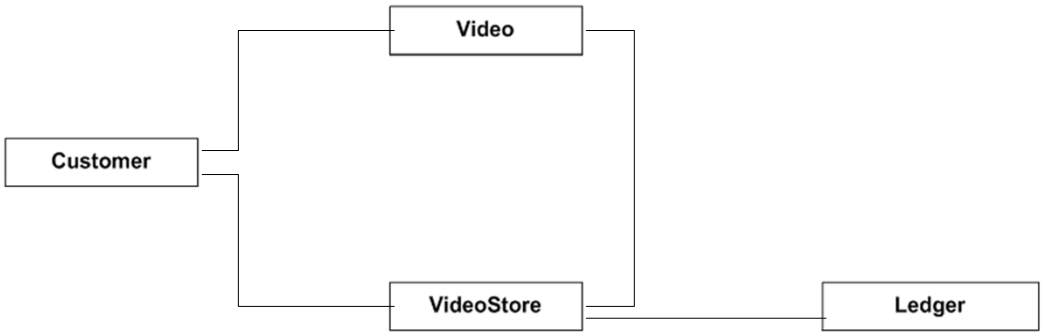
Step 1: Find the *conceptual classes* in the domain

Step 2: Draw the classes in a UML class diagram (or SysML Block Diagram)

Step 3: Identify associations and attributes between conceptual classes

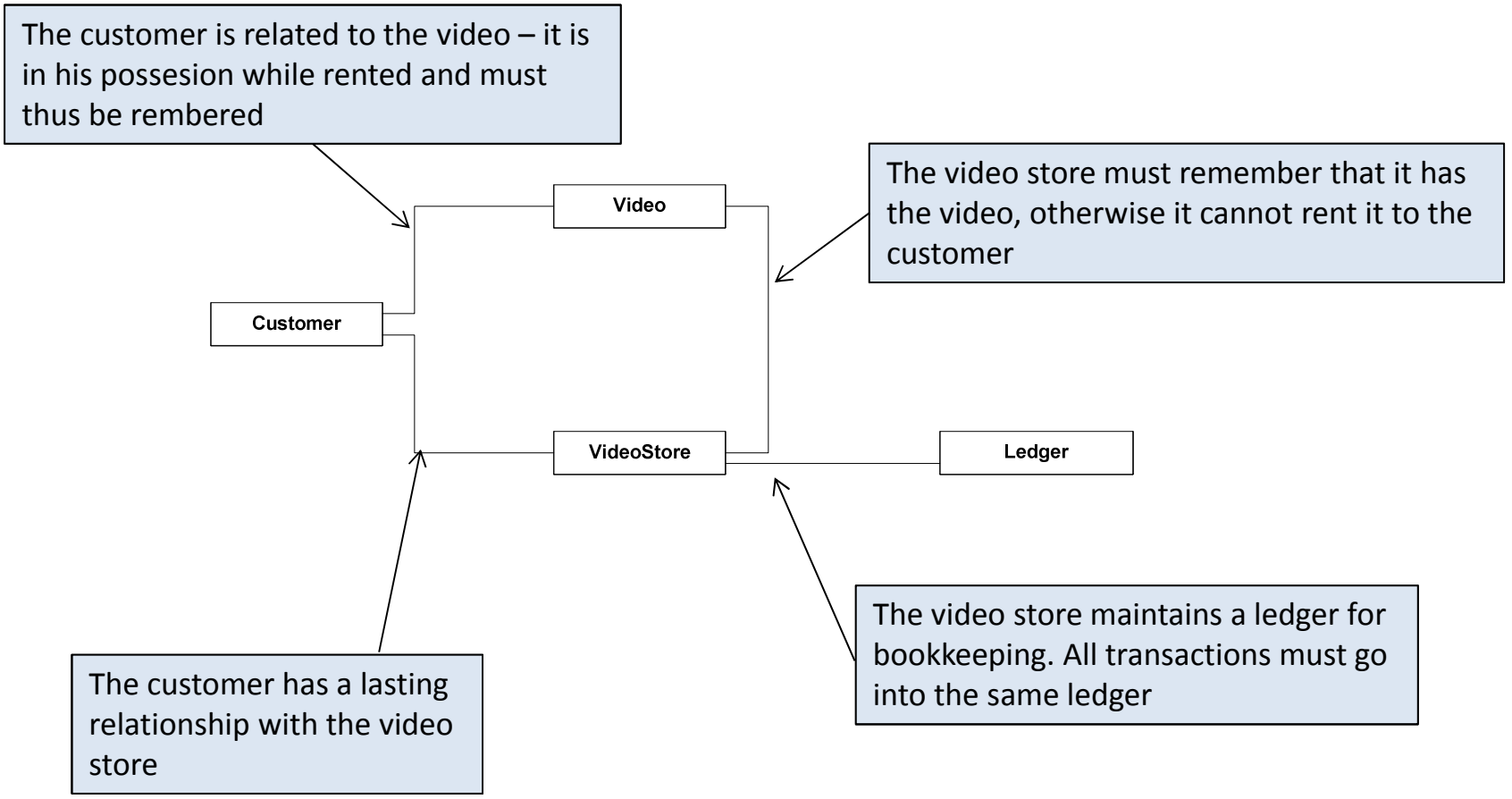
Creating a Domain Model, step 3: Identify associations and attributes

- The conceptual classes identified in Step 1 and drawn in Step 2 can now be associated with each other



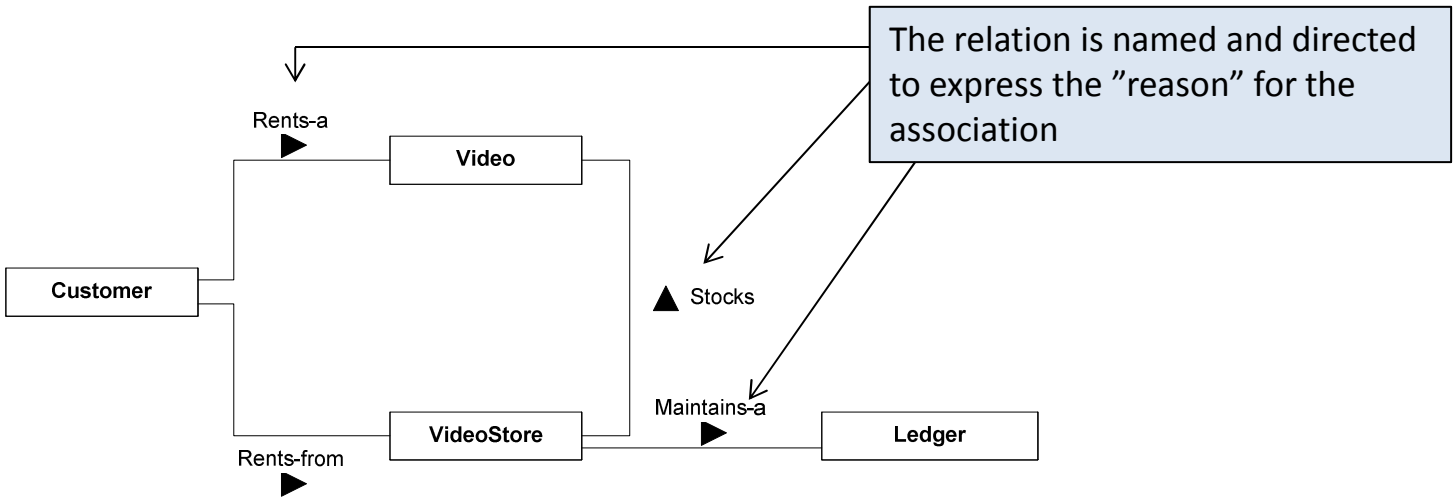
- …but how are associations *identified* and *named*? Some guidelines

Domain Model: Conceptual classes and associations



Associations: Naming

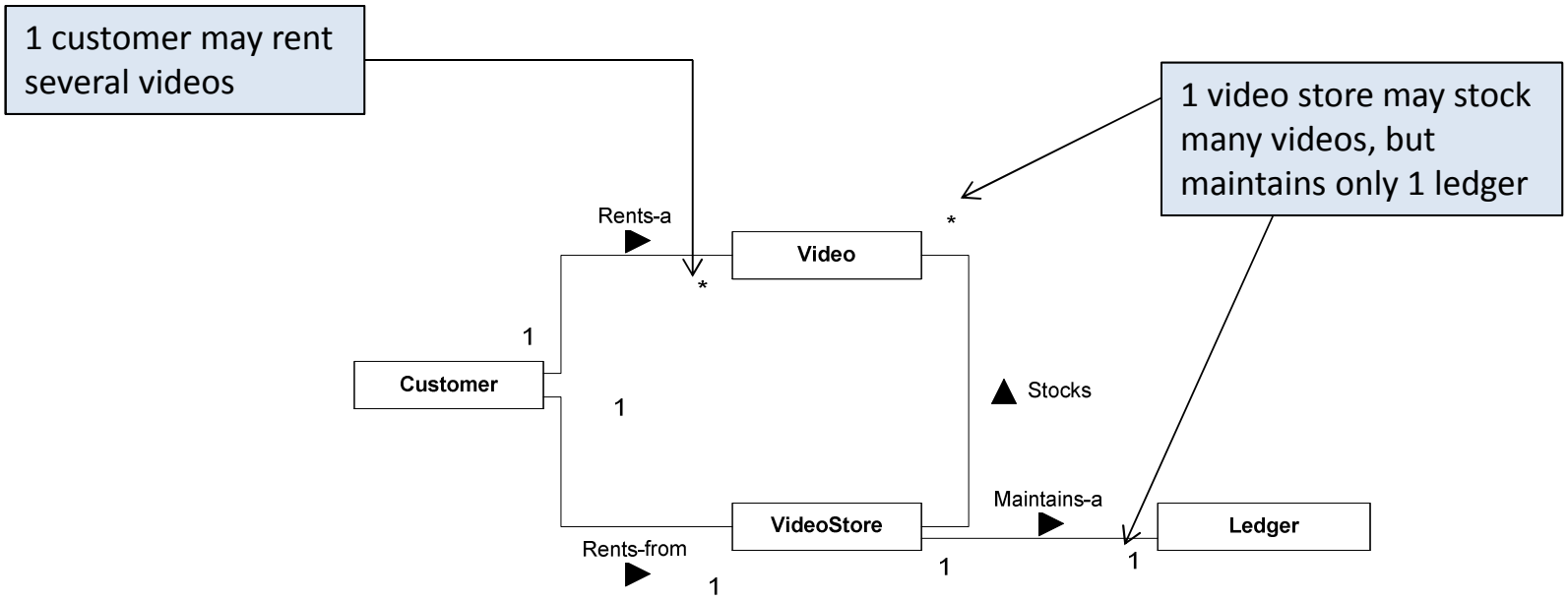
- Associations can be *named* and *directed* to further enhance the meaning and expression power of the DM



- Again: The arrow is only there to aid the understanding of the model – it implies *nothing* in terms of HW or SW association

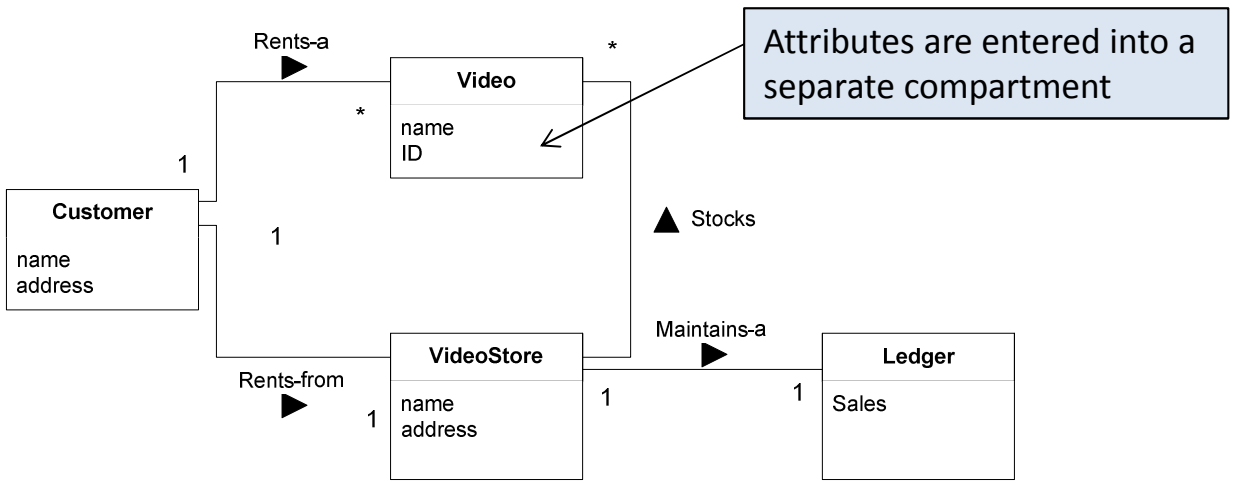
Associations: Multiplicity

- Associations can be assigned multiplicity

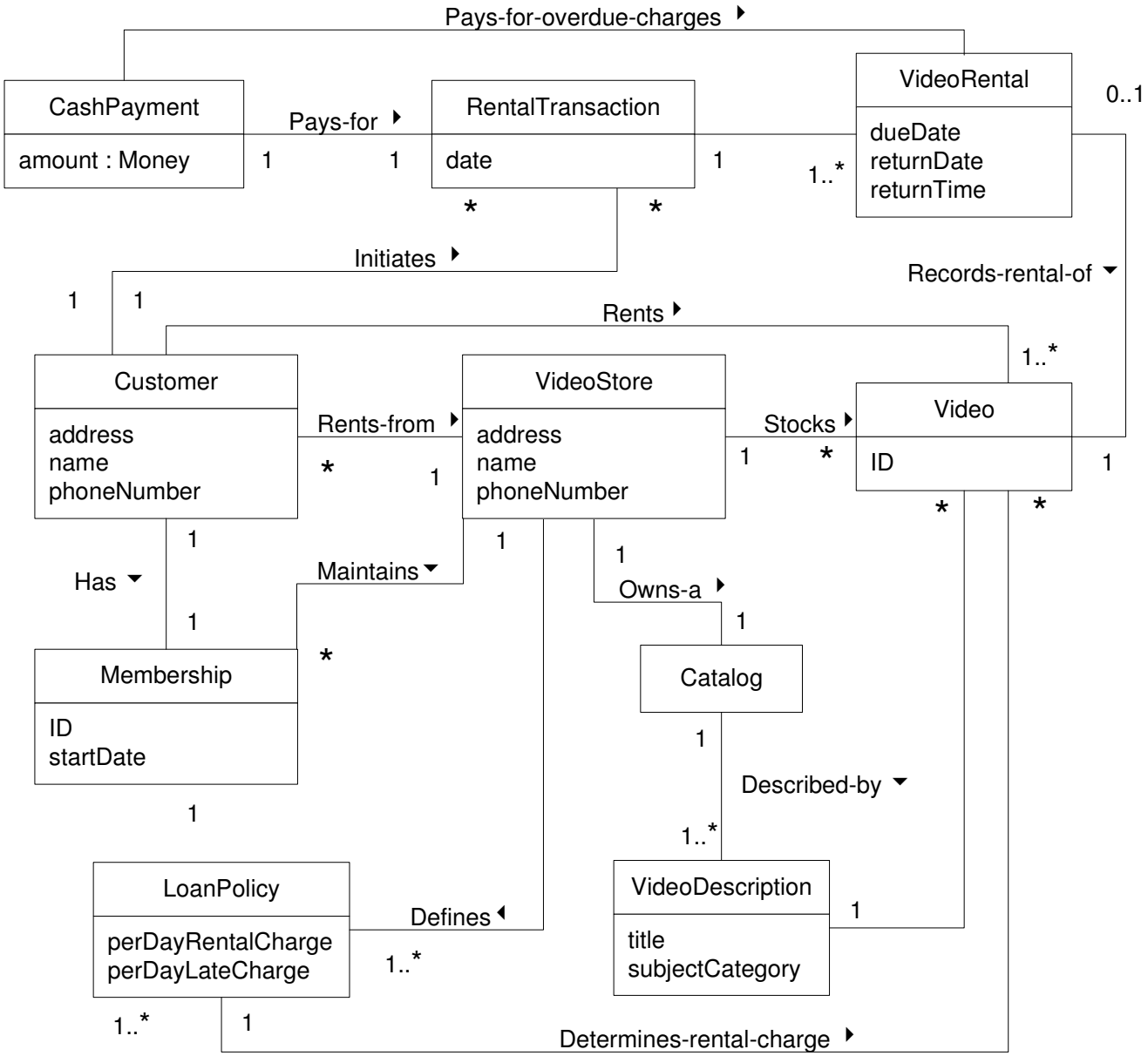


Identifying attributes

- The conceptual classes may also have attributes
- Attribute: A *logical data value* of the class needed to satisfy the currently investigated requirements



EXAMPLE: Video Rental



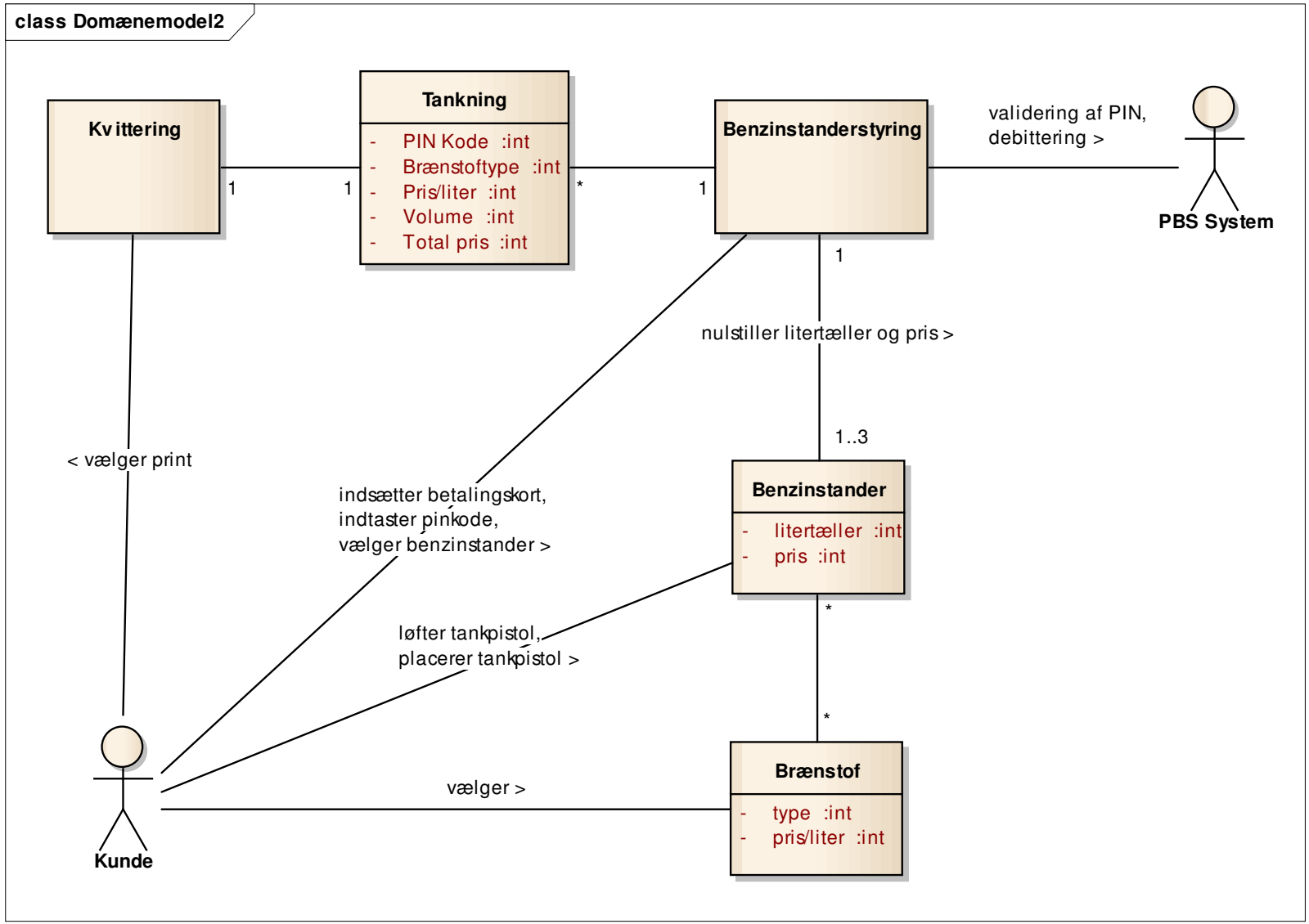
Notice how this can be viewed as a "visual dictionary." It *illustrates* concepts, words, things in a domain.

Exercise for next session: *Service station*

- Create a domain model for the service station based on the use case ”Optank Bil” (see text on CN):

1. Identify meaningful conceptual classes using the methods you have learned about
2. Create a UML Class Diagram with the conceptual classes
3. Create and name associations between classes
4. Set multiplicities where applicable
5. Add attributes to the classes

Domænemodel ”Optank Bil”



System Application Models

I2ISE

UCs are important!

Application models – bridging the gap

- A lot of time has been spent on writing use cases and making domain models. Today, we cash in!
- We will use the UCs to bridge the gap between *what* the system must do (requirements) and *how* it must be done (design)
- In other words, we will use the UC's as *design drivers*
- So it would seem that :

UCs are important!

The System Application Model

- The application model is a first, *incomplete* shot of a design – the ”bridge”

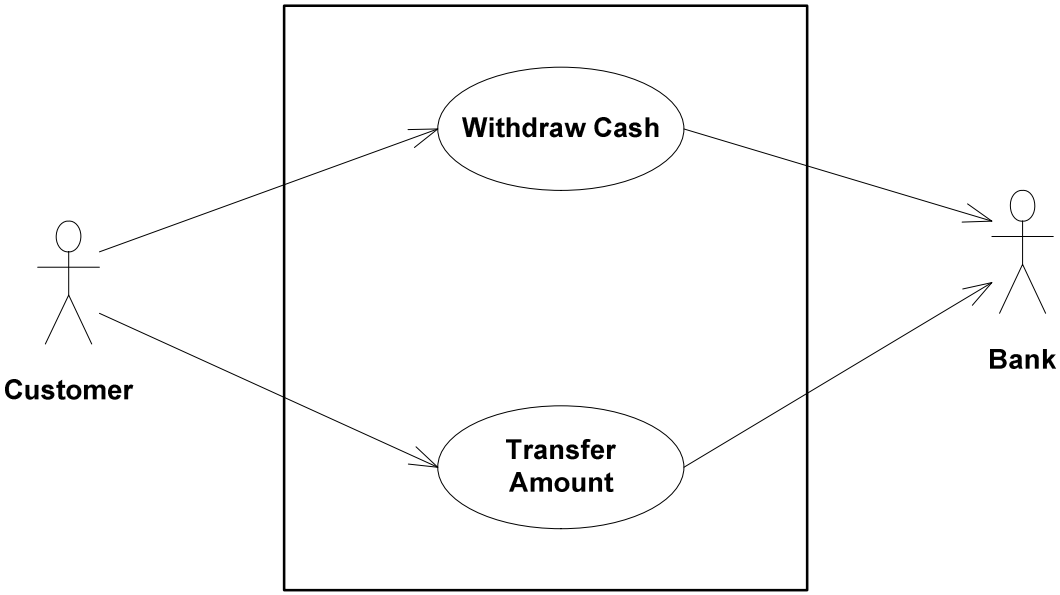
- The application model is based on the system’s *use cases* and the *domain model*.
  - So, again:

UCs are important!

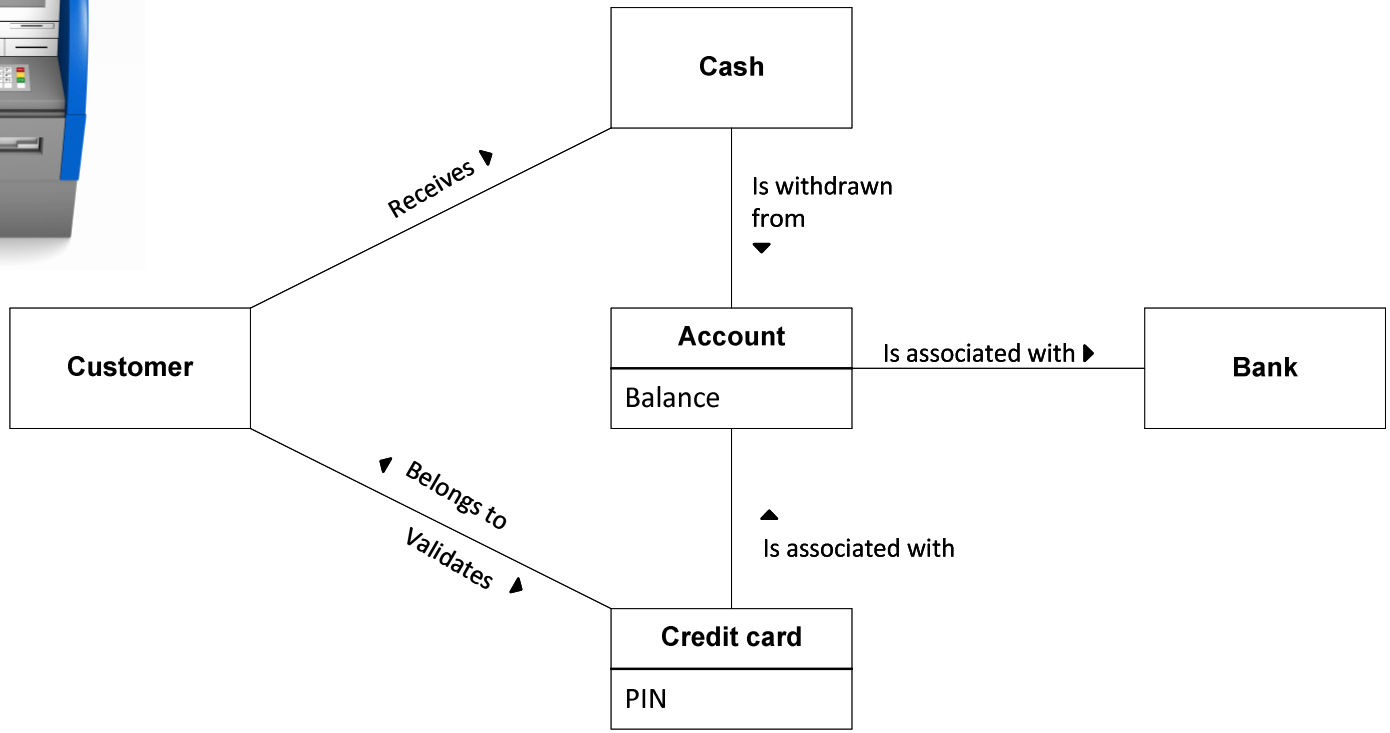
- The application model is built using three different types of diagrams
  - *Class diagrams* for structure
  - *Sequence diagrams* and *state machine diagrams* for behaviour

Today’s example: The ATM

ATM use cases



ATM domain model



The System Application Model – Step 1

- The application model is constructed incrementally in units of use cases. So, apparently,

UCs are important!

Step 1.1: Select the next fully-dressed UC's to design for (how?)

Step 1.2: Identify all actors involved in the UC → *Boundary* classes

Step 1.3: Identify relevant classes in the domain model involved in the UC → *Domain* classes

Step 1.4: Identify the UC *controller* → *Controller* class

Identify the what, the what and the what?!?

- Our application model consists of three different types of classes: *Boundary*, *domain*, and *controller* classes

- *Boundary* classes represent UC *actors*
  - They are the actors' interface to the system (UI, protocol, …)
  - They *present* the system but contain no business logic.
  - 1 per actor, shared between UCs
  - Optionally stereotyped «boundary»

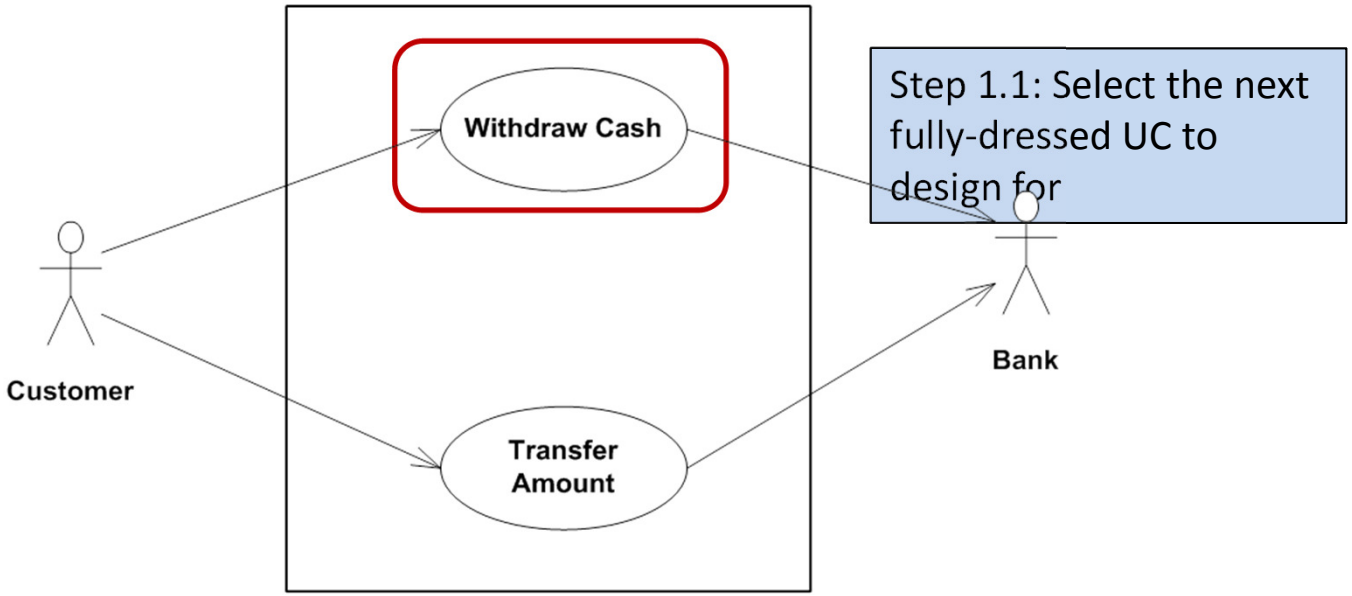
- *Domain* classes represent the system's *domain*
  - Memory, domain-specific knowledge, configuration, etc.
  - 1 or more, shared between several UCs

Identify the what, the what and the what?!?

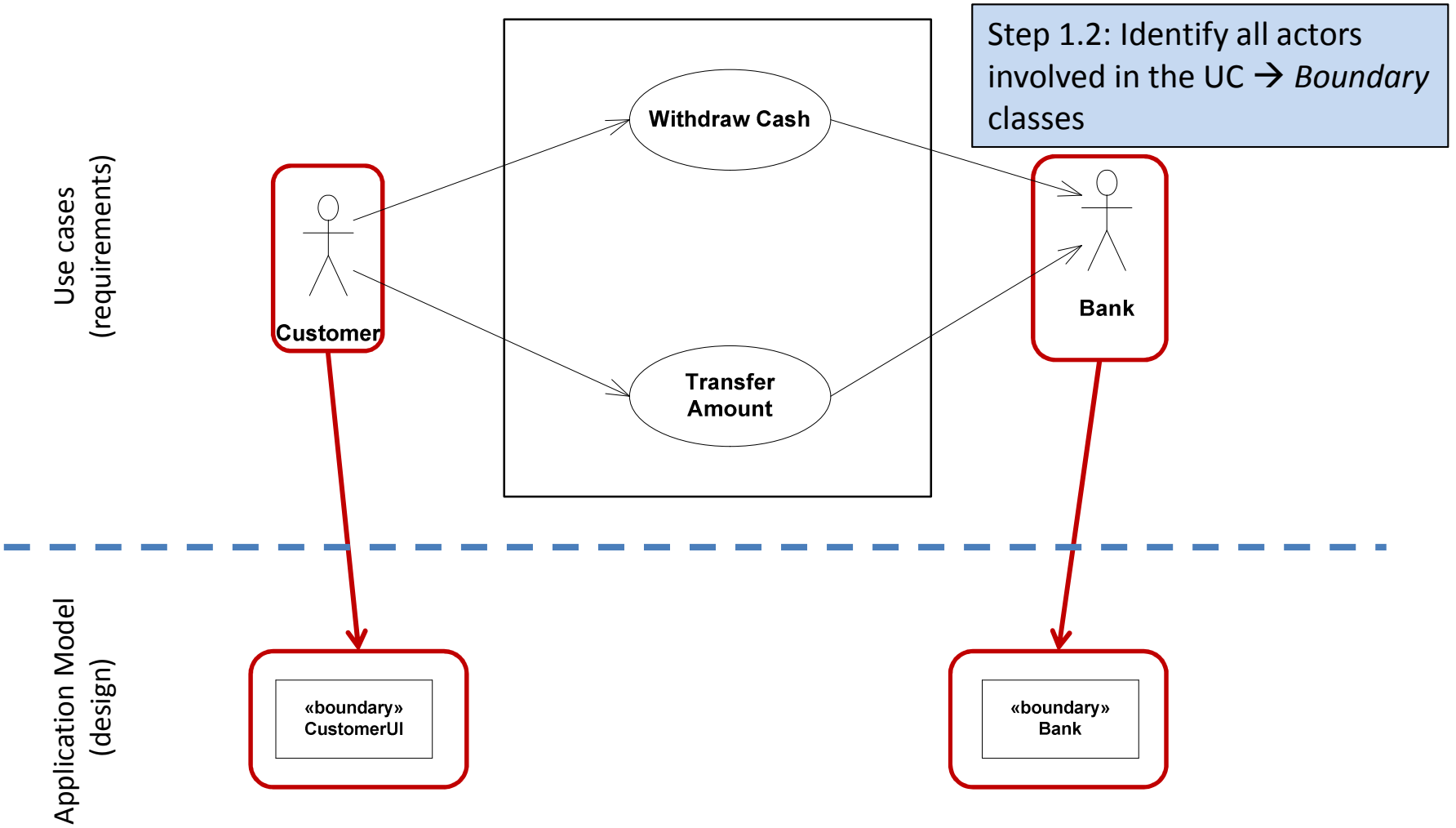
- Our application model consists of three different types of classes: *Boundary*, *domain*, and *controller* classes

- The *Controller* class holds the UC business logic
  - It ”executes” the use case by interacting with the boundary and domain classes.
  - Named after the UC
  - Typically 1 per UC or 1 shared among a couple of UCs
  - Optionally stereotyped «`control`» or «`controller`»

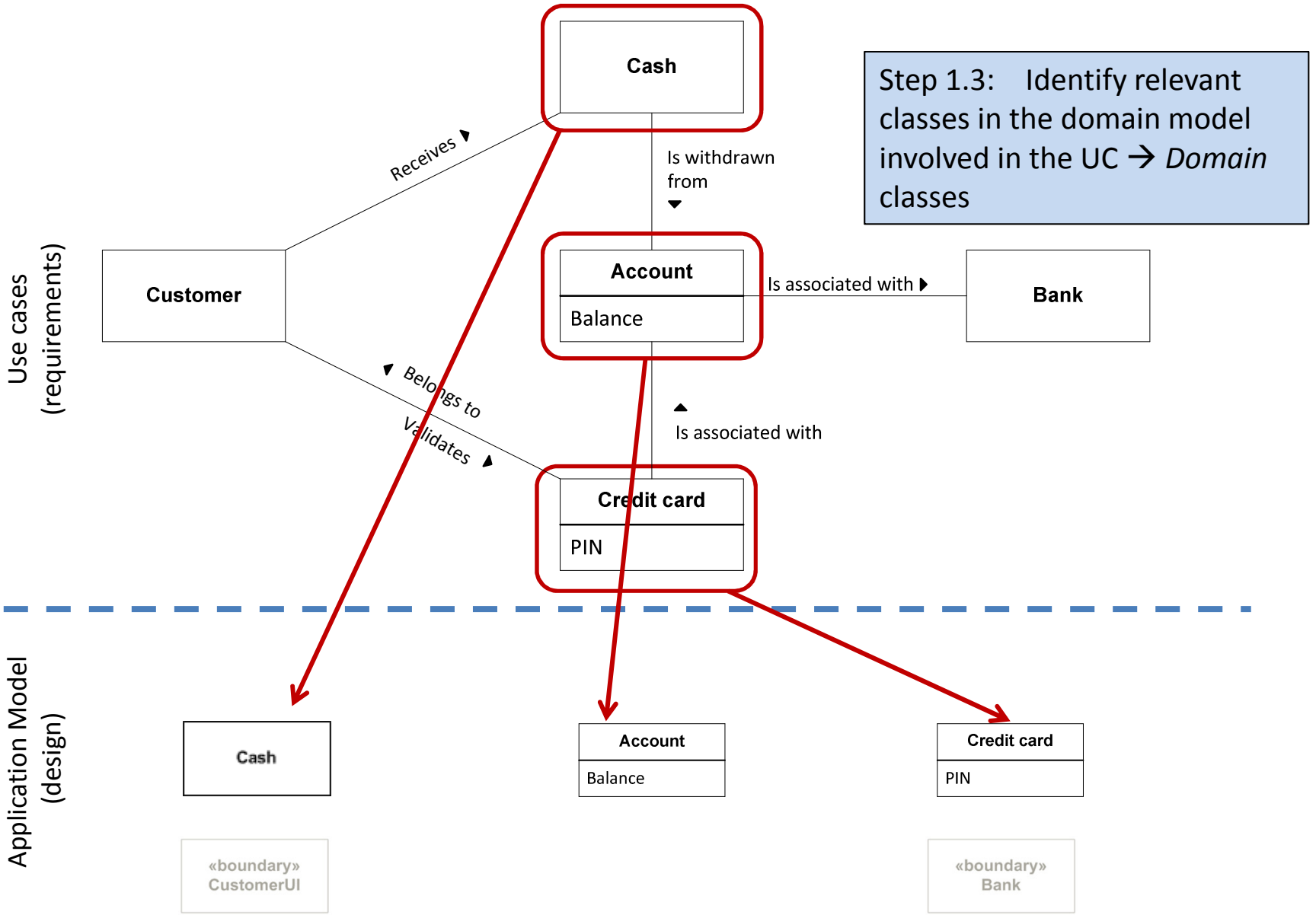
ATM step 1.1: Select next Use Case



ATM step 1.2: Actors -> boundary classes

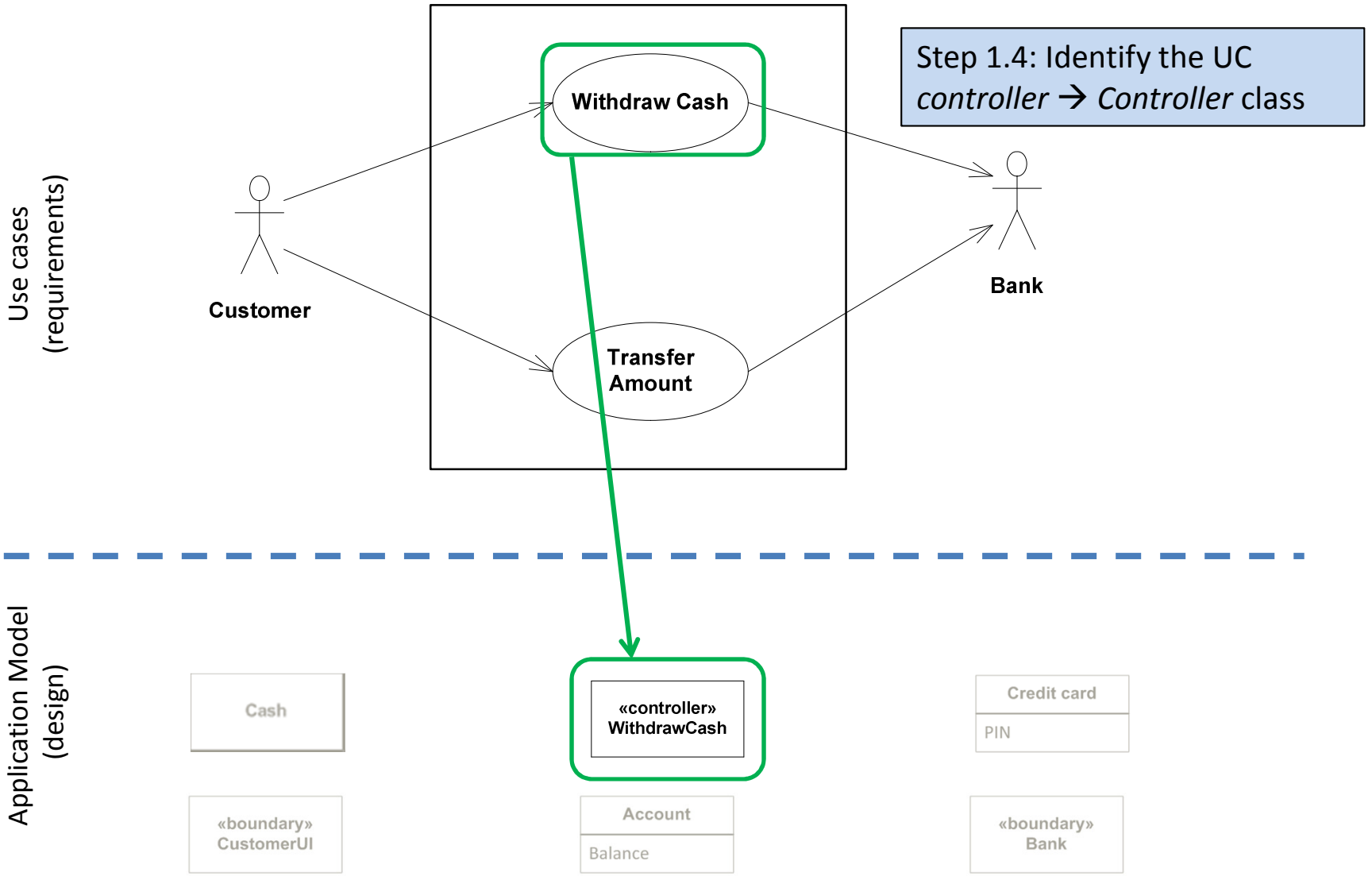


ATM step 1.3: Domain classes



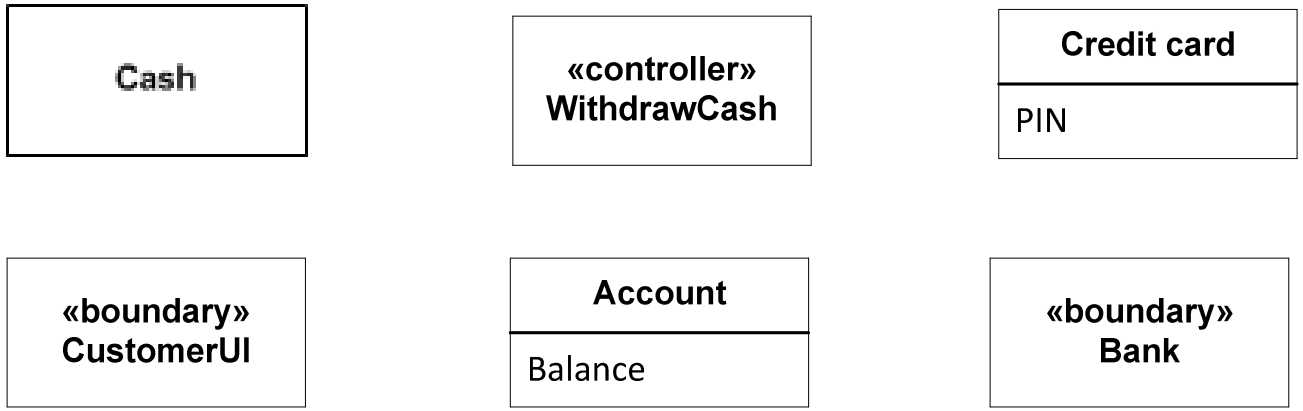
Step 1.3: Identify relevant classes in the domain model involved in the UC → *Domain* classes

ATM step 1.4
Identify UC controller -> Controller class



Step 1 complete – so far, so good

- We have now completed Step 1 and identified 6 candidate SW classes for our initial design
- To do this, we used our *use cases* and our *domain model*



- We must now add *behaviour* to these classes – that's Step 2

The System Application Model – Step 2

- The collaboration between the classes is now explored from the UC description – so, still,

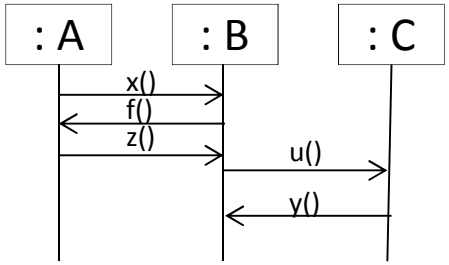
UCs are important!

Step 2.1:	Go through the UC main scenario step-by-step and identify collaborations (actor- or system-initiated)
Step 2.2:	Update the application model's sequence and class diagrams to reflect the collaboration (relations, operations, attributes)
Step 2.3:	Identify any classes with state-based behavior and update STMs for the classes (states, events, transitions) . *(Step 2.3 is skipped if none classes with state-based behavior)*
Step 2.4:	Verify that the diagrams adhere to the UC (descriptions, test)
Step 2.5:	Repeat 2.1 – 2.3 for all UC exceptions. Refine model.

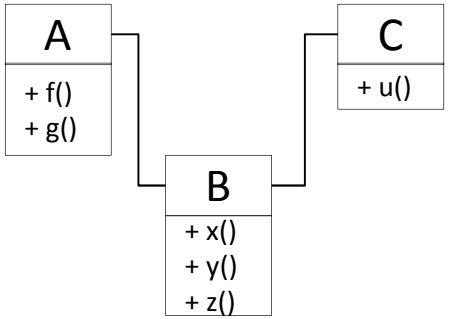
- Note: All 3 diagrams (class, SEQ, STM) are updated at the *same time* in this process.

Principle for step 2: Go through main scenario, update collaborations

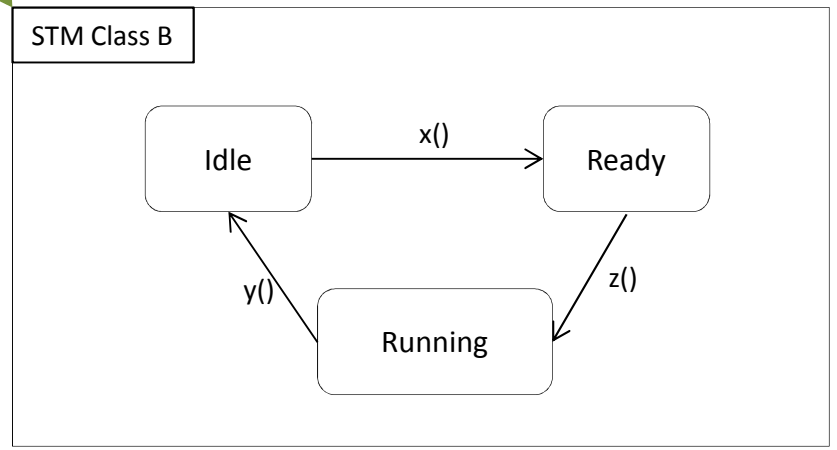
Sequence diagram:
- Add sequence of calls to objects



Class diagram:
- Add operations to classes
- Update associations



Updated simulta-neously

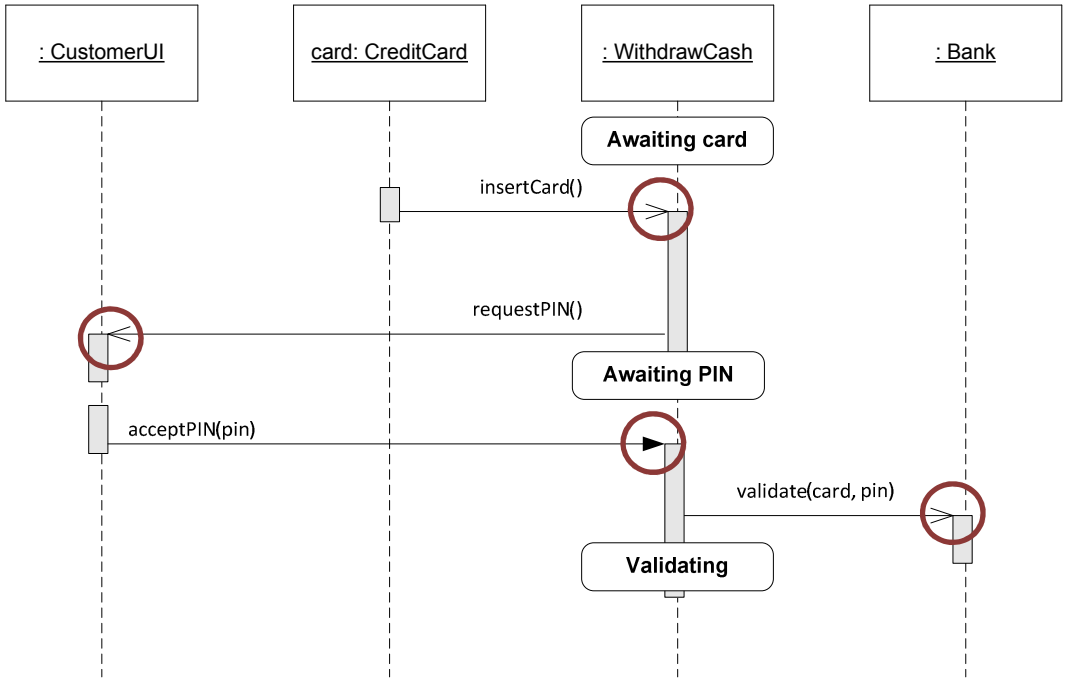


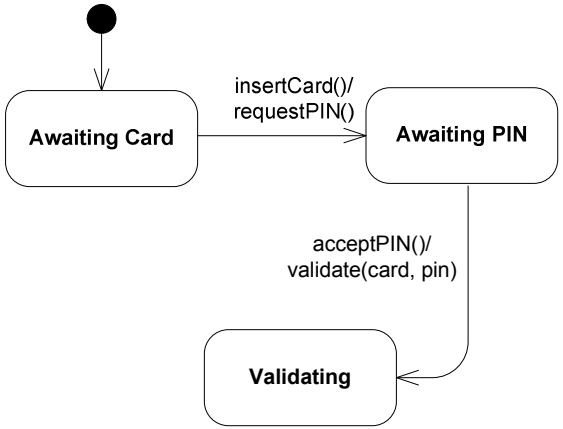
State diagrams:
- Add states to stateful classes
- Add actions (operations) that changes the state

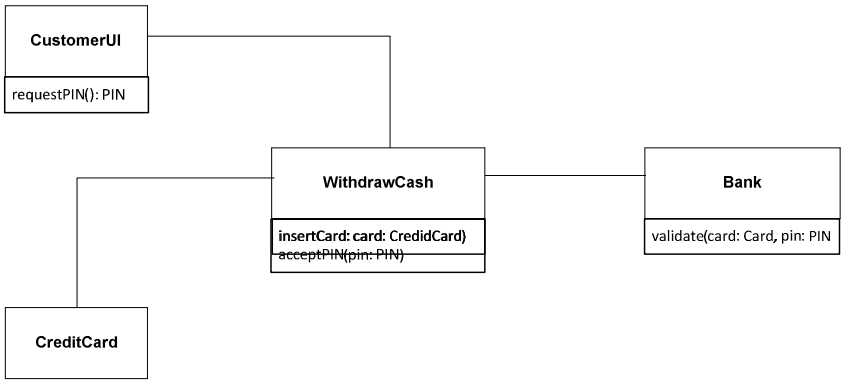
Steps 2.1-2.4 for UC *Withdraw Money*

Main scenario:

1. Customer inserts credit card in System
2. System requests Customer's PIN code
3. Customer enters PIN code
4. System validates card info and PIN code with Bank







Your turn: Complete system architecture model for UC *Withdraw Cash*

- The complete text for UC Withdraw Cash is on the CampusNet. You have the following tasks:
  - Complete the System Architecture Model for the main scenario for the UC
  - Complete the System Architecture Model for all extensions for the UC

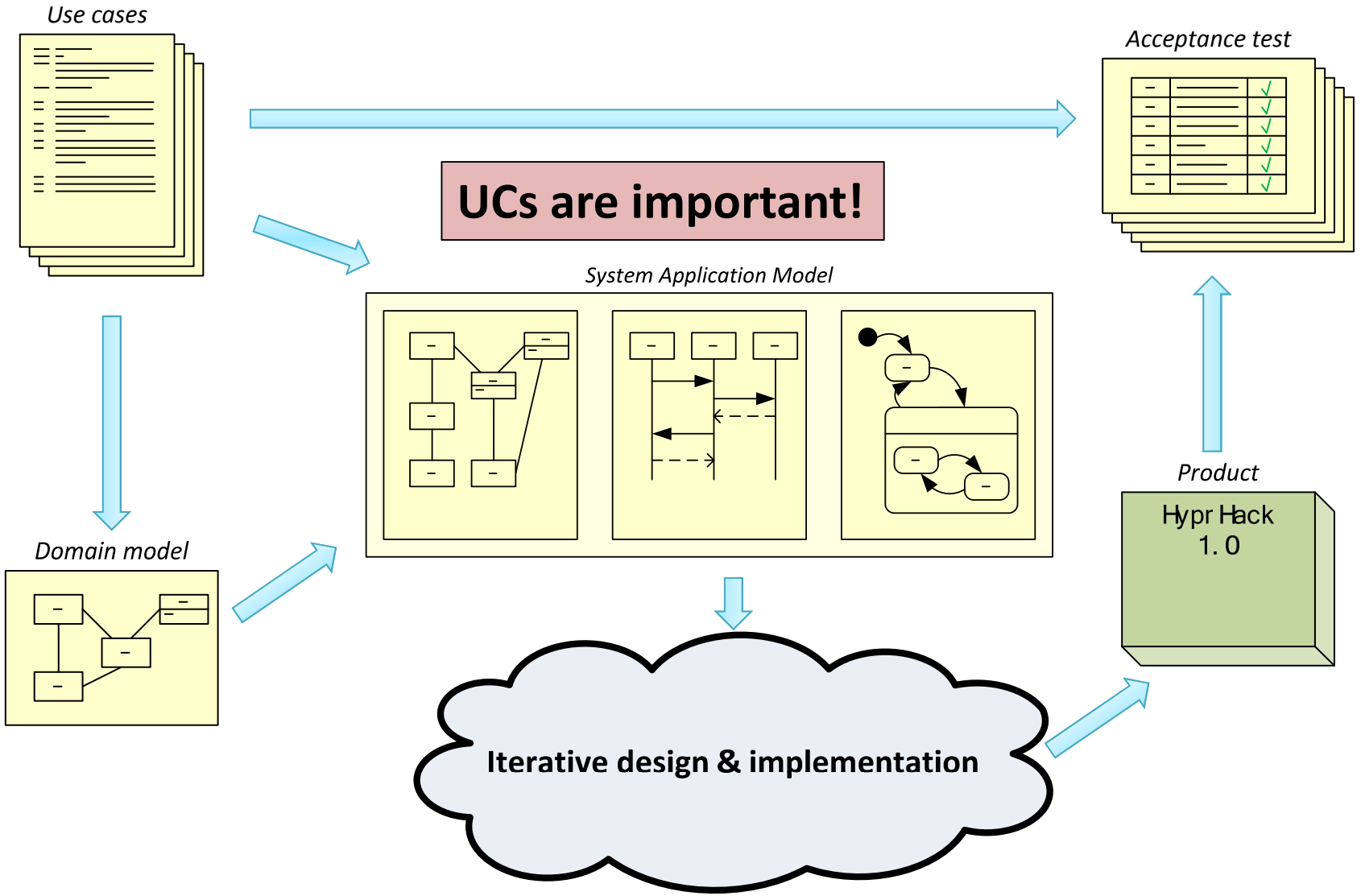
- ***SAM_ATM UC description.pdf***
- ***SAM_ATM_UC Withdraw Cash Solution.pdf***

The System Application Model – Step 3 and beyond

- As you add more UCs to the application model you will begin to discover *reuse* of the previous classes
  - Domain and boundary classes often repeat
  - Different domain classes may be so closely related that they might as well be ”collapsed” into one
  - Sometimes, even controllers ”collapse”

- At this time, experience must ensure the correct cut between reuse and new classes

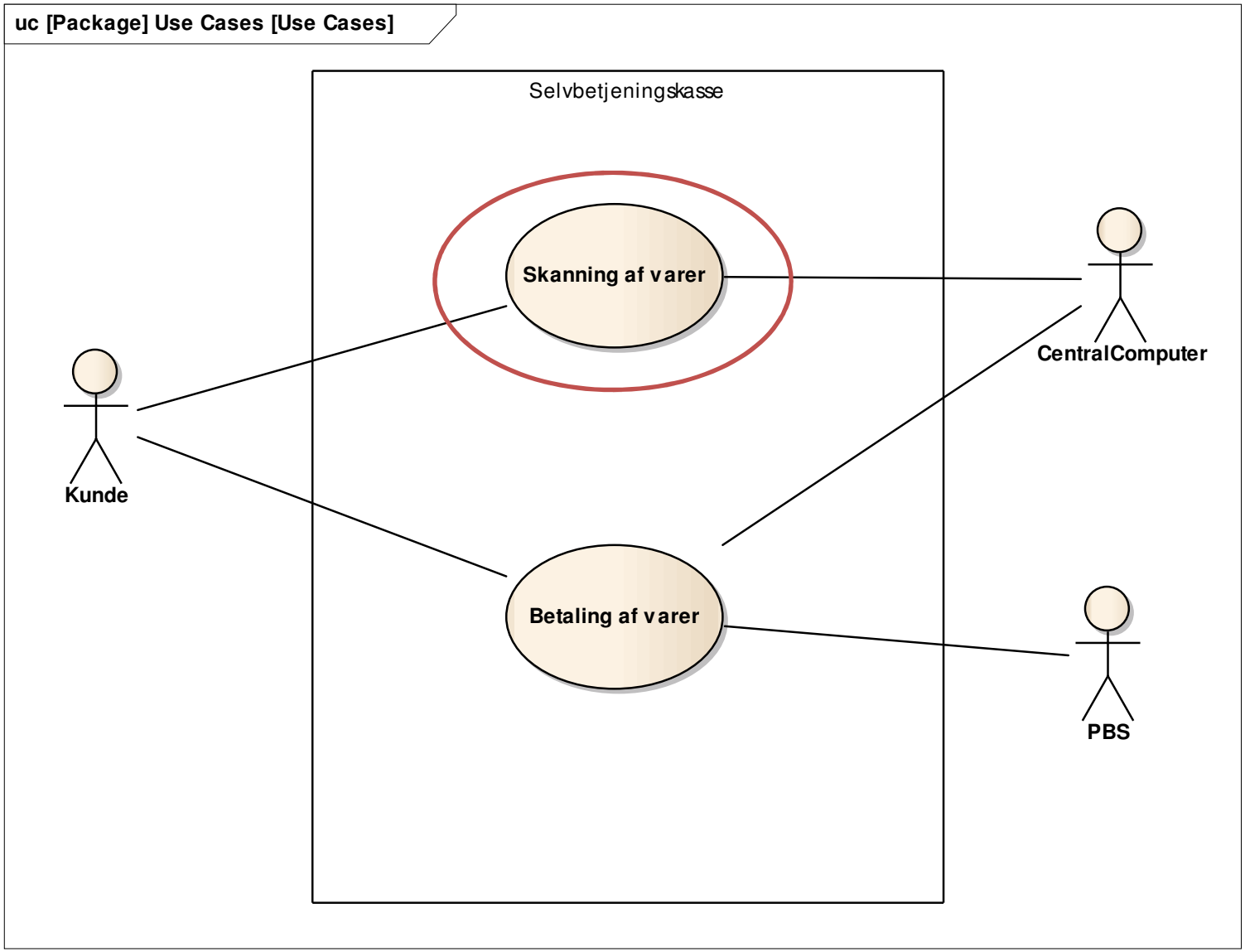
UCs in the big picture (1 iteration)



Application model and subsystems

- In case a system to be developed is composed by more computers an **application model is create for each subsystem** (computer)
- **Boundary classes** are identified for connections **between the subsystem and external units and actors**
- **Controller classes** are identified for **Use Cases where the subsystem is involved**
- **Domain classes** are taken from the domain model

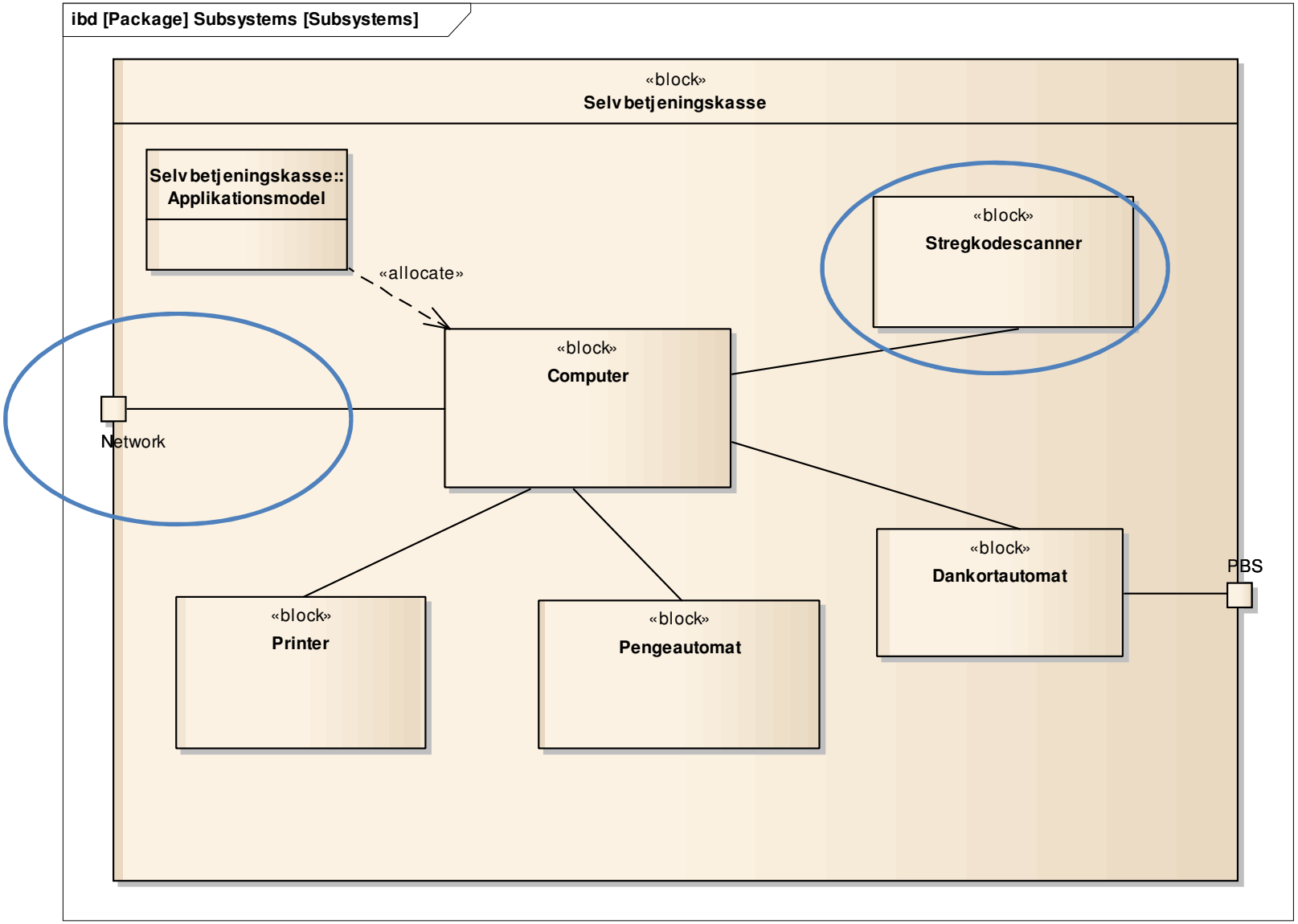
Selvbetjeningskasse (2. eksempel)



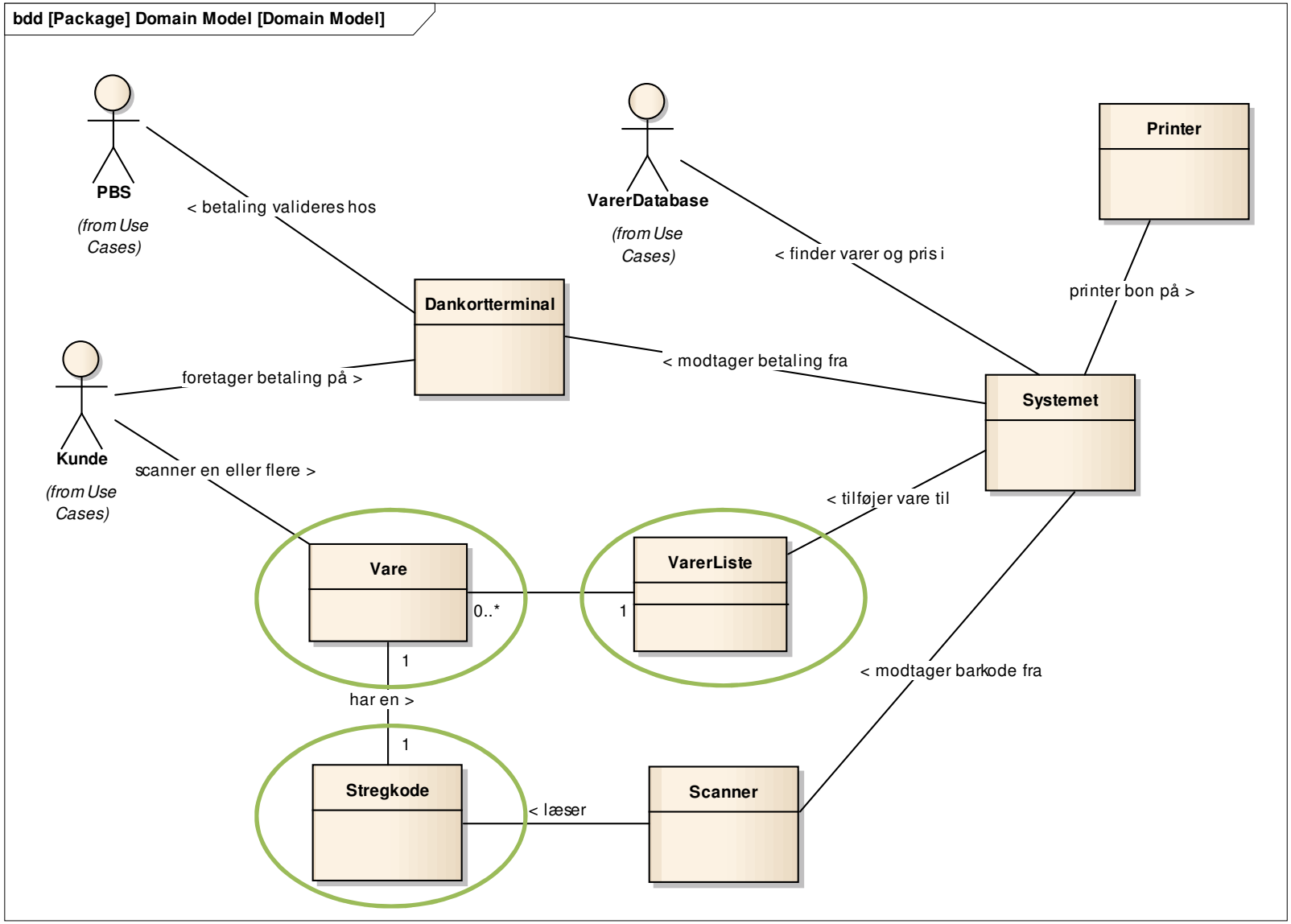
Scanning af Vare (Hovedscenarie)

1. Selvbetjeningskassen anmoder kunden om at skanne vare
2. Kunden placerer vare foran skanner
3. Systemet skanner varens stregkode
4. Systemet finder varens pris i varedatabasen
5. Vare med pris tilføjes til en vareliste
6. Kunden lægger vare i pose på bordet ved siden af skanner
7. Punkterne 1-6 gentages indtil alle varer er skannet
8. Kunden vælger afslut

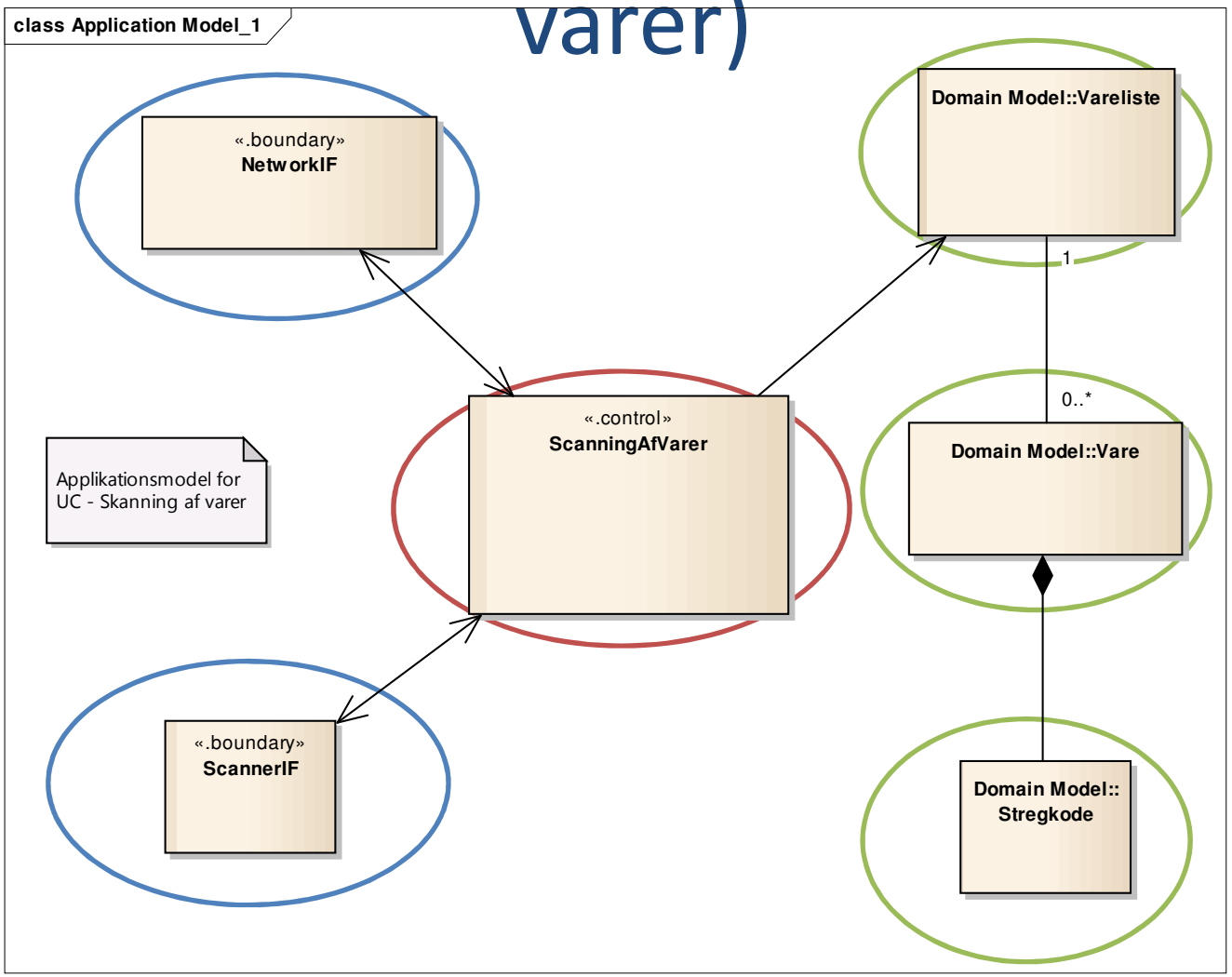
Allokering – IBD Subsystem - Boundary



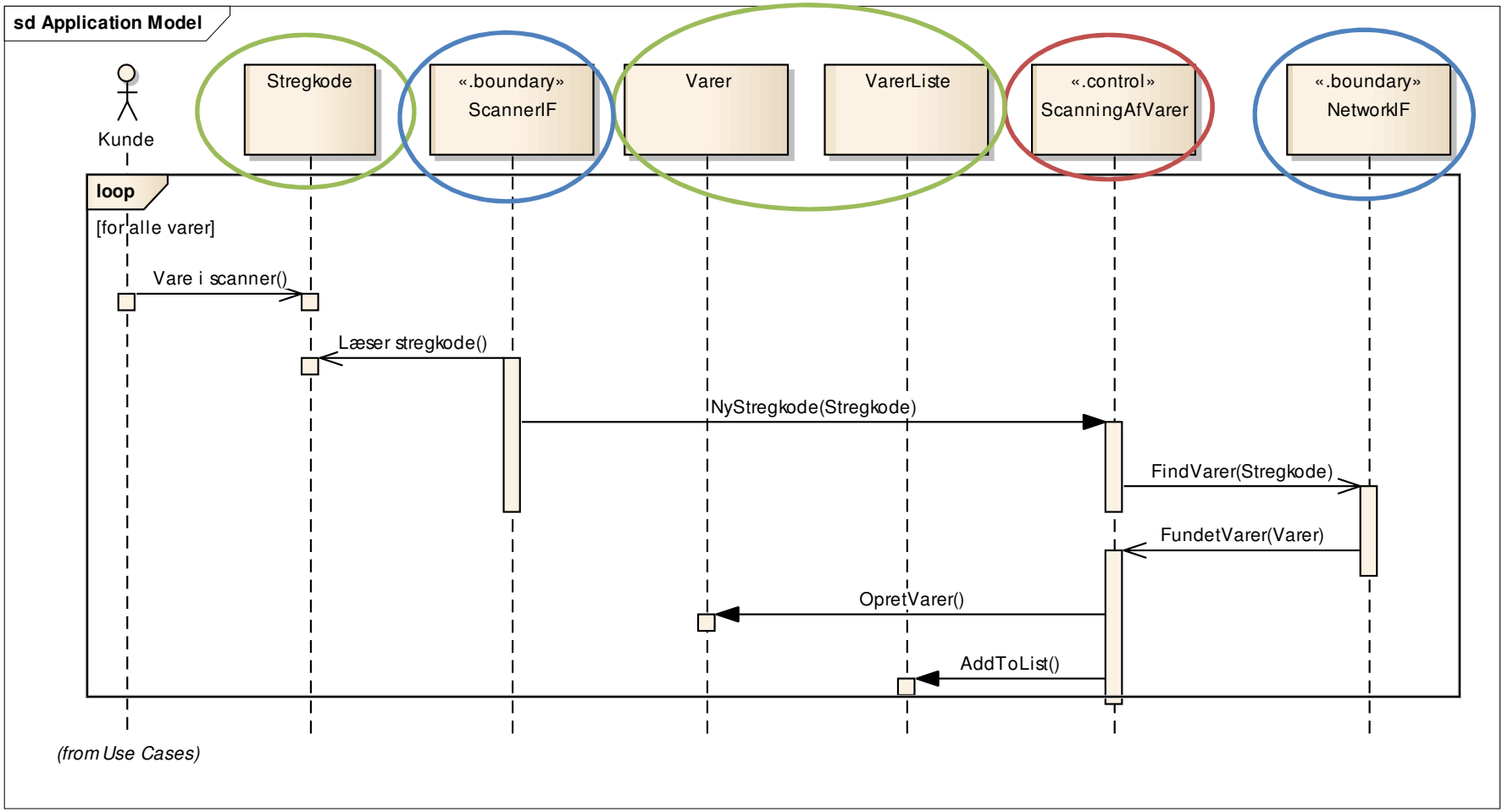
Domain model



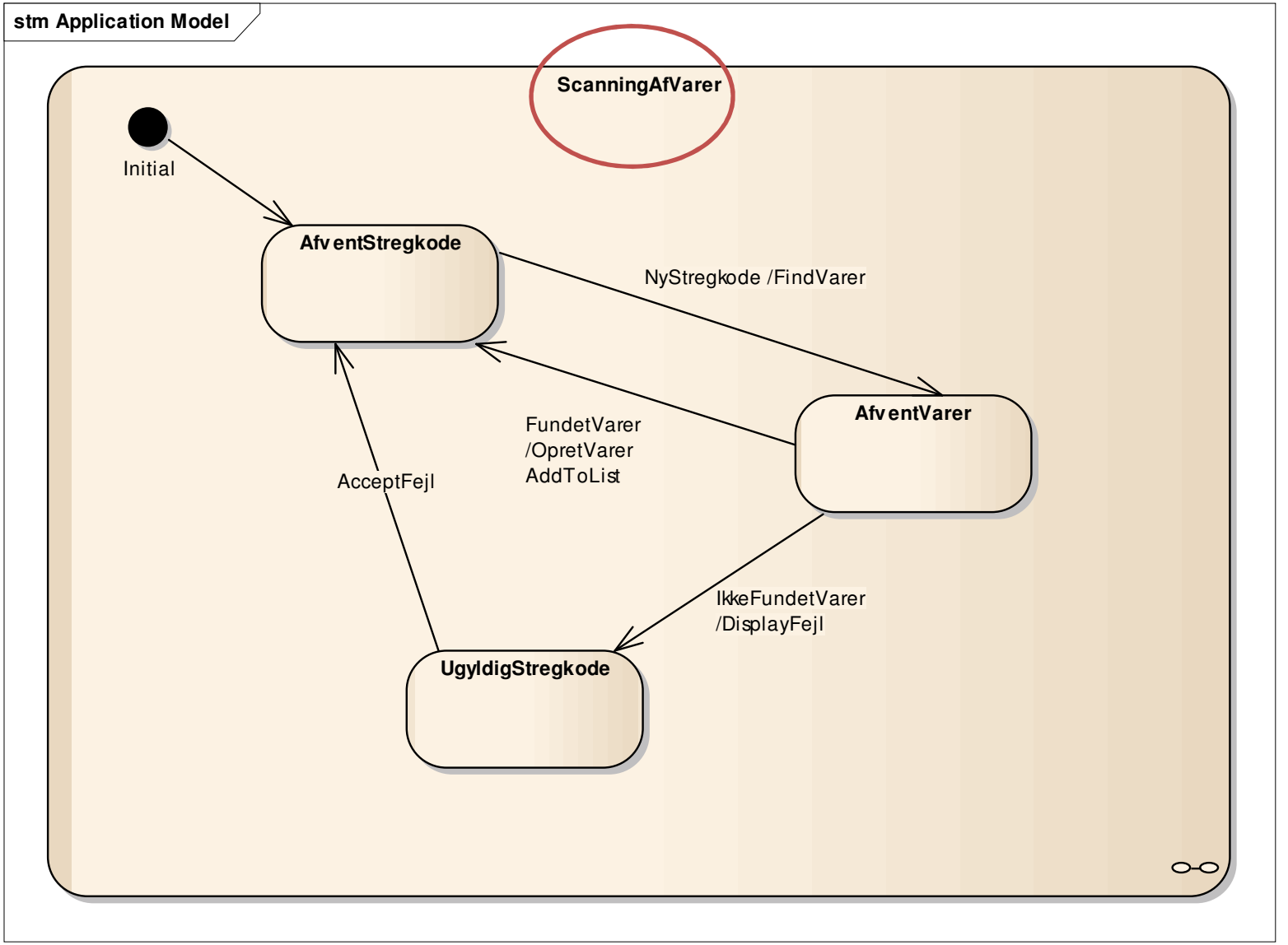
Applikationsmodel (UC – Scanning af varer)



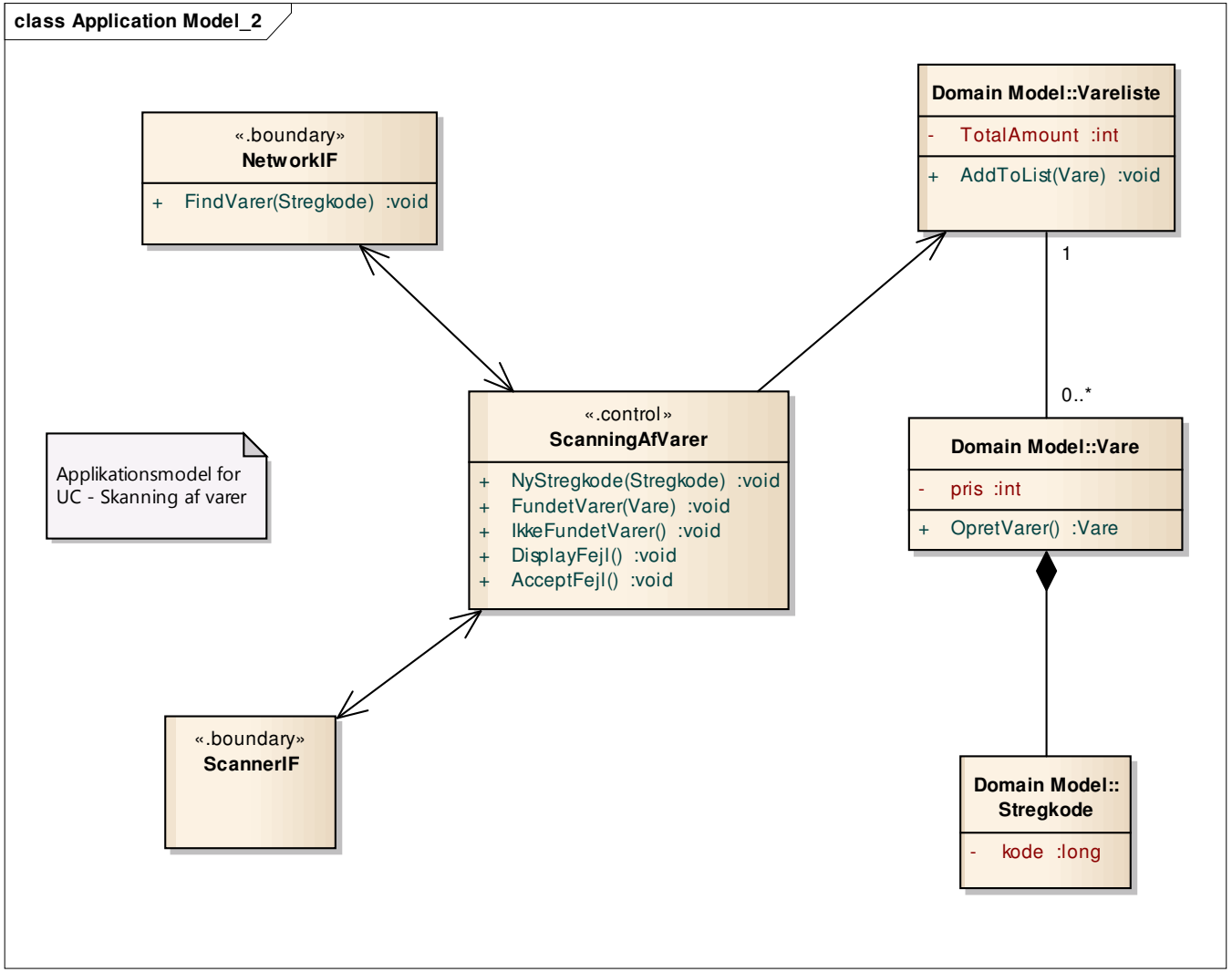
Sekvensdiagram (Applikationsmodel)



Statediagram (Control class)



Opdateret applikationsmodel



C-Koden

```
class ScanningAfVarer
{

public:
    ScanningAfVarer();
    virtual ~ScanningAfVarer();
    NetworkIF *m_NetworkIF;
    ScannerIF *m_ScannerIF;
    VarerListe *m_VarerListe;
    Stregkode *m_Stregkode;

    void NyStregkode(Stregkode stregkode);
    void FundetVarer(Varer varer);
    void IkkeFundetVarer();
    void DisplayFejl();
    void AcceptFejl();

};
```

```
class NetworkIF
{

public:
    NetworkIF();
    virtual ~NetworkIF();
    ScanningAfVarer *m_ScanningAfVarer;

    void FindVarer(Stregkode stregkode);

};
```

```
class VarerListe
{

public:
    VarerListe();
    virtual ~VarerListe();
    Varer *m_Varer;

    void AddToList(Varer varer);

private:
    int TotalAmount;

};
```

```
class Varer
{

public:
    Varer();
    virtual ~Varer();
    Stregkode *m_Stregkode;

    Varer OpretVarer();

};
```

Er designet komplet?

Hvad mangler?