Applikationsmodeller
Part 3
Systemer med subsystemer

I2ISE

Dagens emner

- Resume regler og guide lines
- Applikationsmodeller for systemer med subsystemer

Kommunikationsregler

- Det er *control* klassen der tager alle logiske beslutninger – derfor gælder:
  - *Boundary* klasser kalder **KUN** til *control* klassen/r!
    - (og evt. nødvendige *domain* klasser der bruges som parametre (*Data Transfer Objects - DTO*))!
  - *Boundary* klasser kalder **IKKE direkte** til andre *boundary* klasser!
    - (medmindre man har en lagdelt *boundary* struktur)!
  - *Domain* klasser kalder **IKKE** til *boundary* klasser!
  - *Domain* klasser kalder **IKKE** noget på eget initiativ!

Kommunikationsregler

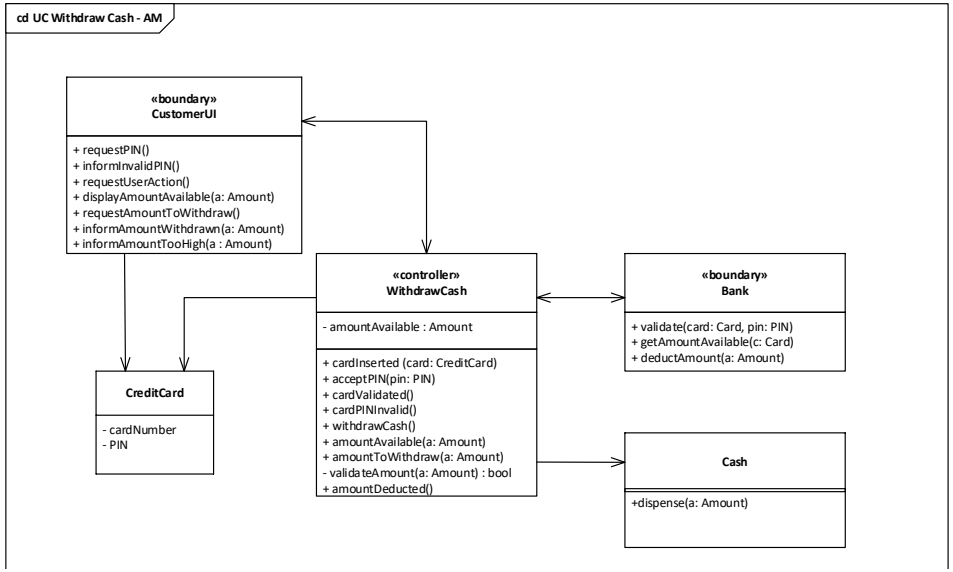
Principielle kommunikationsregler i arkitekturen

	Til Boundary	Til Domain	Til Control
Fra Boundary	Nej!	Nej!	Ja!
Fra Domain	Nej!	Nej!	Nej!
Fra Control	Ja!	Ja!	Ja!

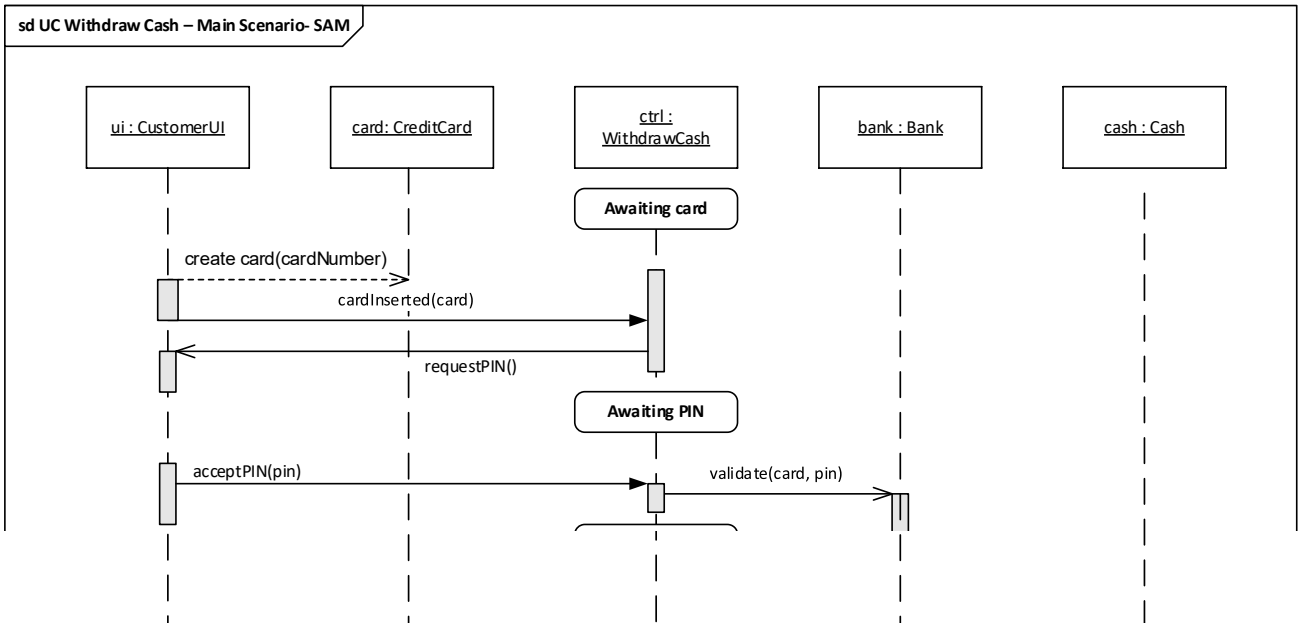
Pragmatiske kommunikationsregler i arkitekturen

	Til Boundary	Til Domain	Til Control
Fra Boundary	(Lagdelt I->I og O->O)	(Data Transfer Object)	Ja!
Fra Domain	Nej!	(Komposition)	Nej!
Fra Control	Ja!	Ja!	Ja!

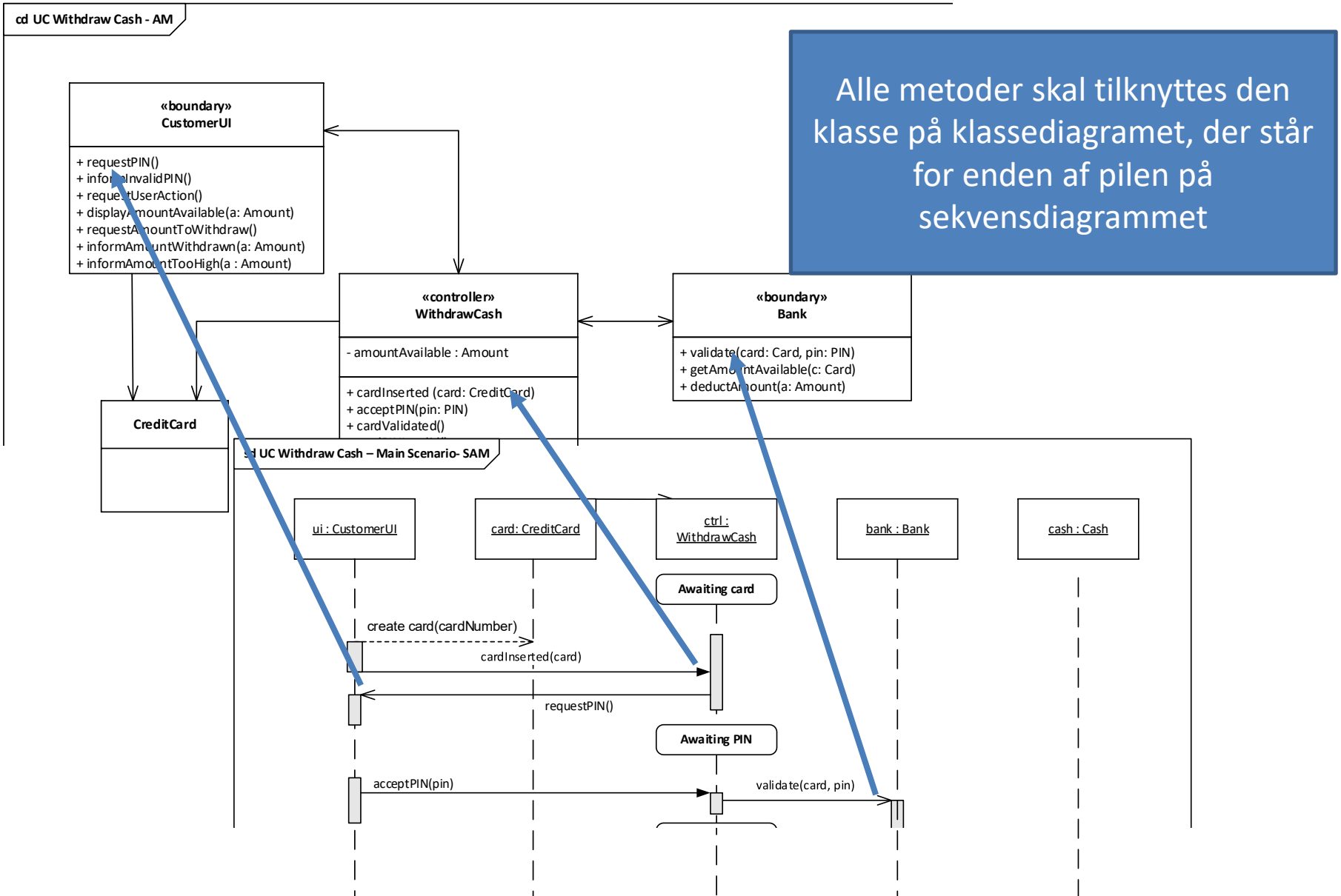
Designregler



Alle klasser på sekvensdiagrammet skal også være på klassediagrammet!

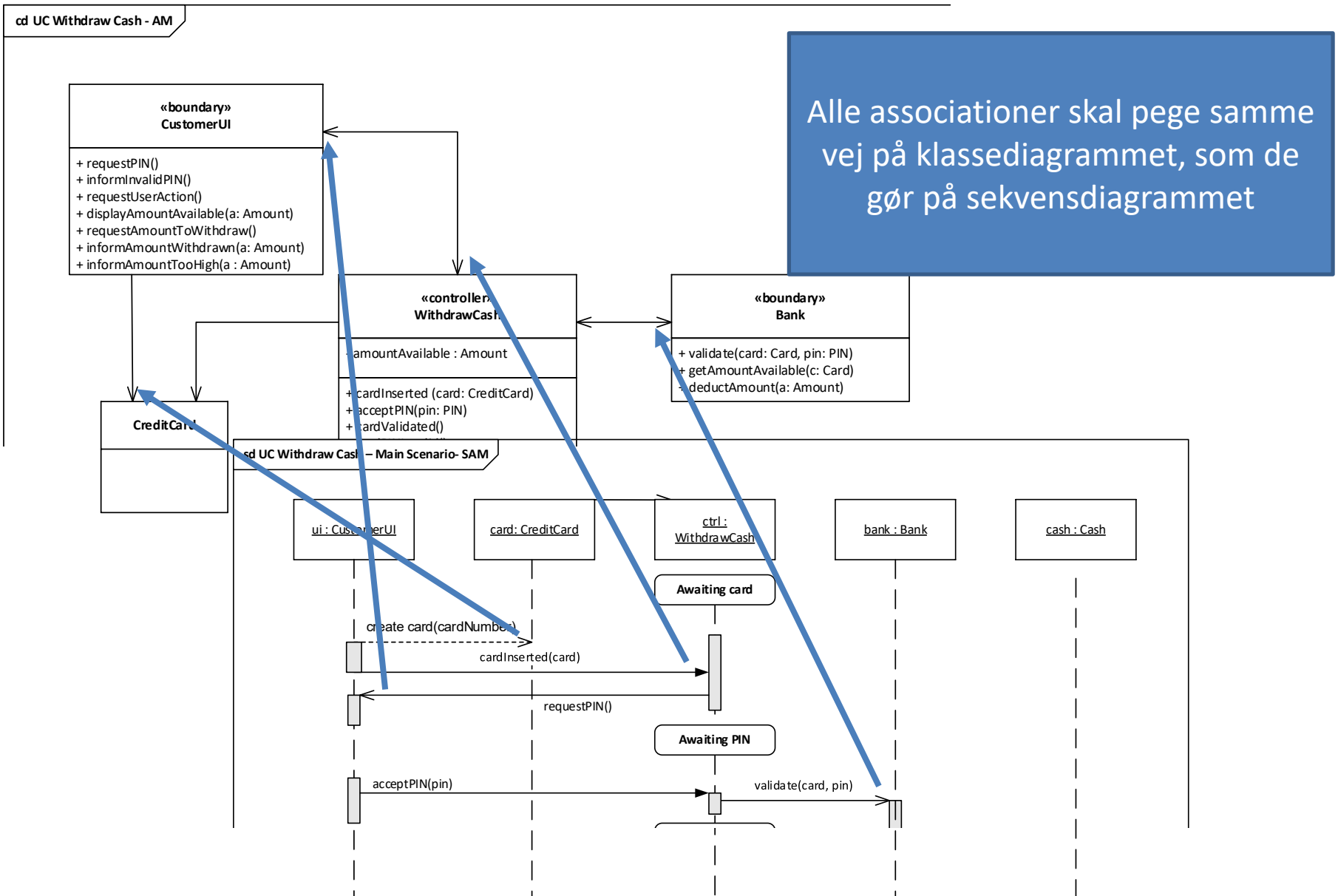


Designregler



Alle metoder skal tilknyttes den klasse på klassediagramet, der står for enden af pilen på sekvensdiagrammet

Designregler



Alle associationer skal pege samme vej på klassediagrammet, som de gør på sekvensdiagrammet

Designregler

- **Alle klasser** på **sekvensdiagrammet**, skal også være **på klassediagrammet** (men ikke nødvendigvis omvendt)
- **Alle metoder** skal tilknyttes **den klasse på klassediagramet**, der står for **enden af pilen på sekvensdiagrammet**
  - Det er et **metodekald** fra den ene klasse til den anden
- **Alle associationer** skal **pege samme vej** på **klassediagrammet**, som de gør på **sekvensdiagrammet**
  - For at den **ene klasse kan kalde den anden**, skal den have **en association til den**
- **Associationer kan være tovejs**, hvis begge klasser kalder den anden i løbet af UC

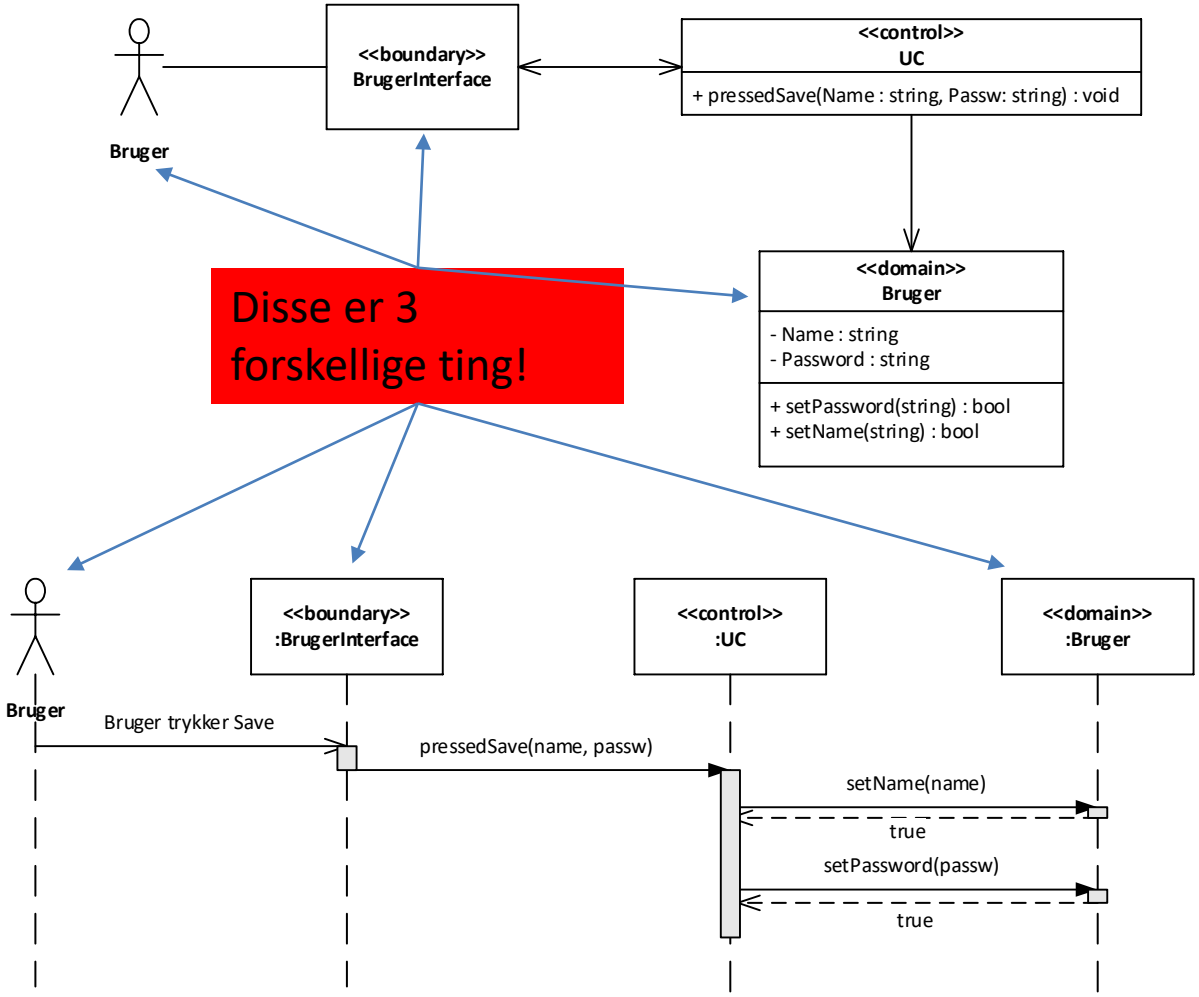
Guidelines "control"

- ***Control* klassen** er **IKKE** det samme som **hardware controlleren** eller **µ-controlleren** eller **”control unit” på BDD/IBD**!
- ***Control* klassen** er en del af **softwaren** som kører **på CPUen** i disse hardware controllers!
- ***Control* klassen** har navn efter den UC, den udfører!

Guideline Actor

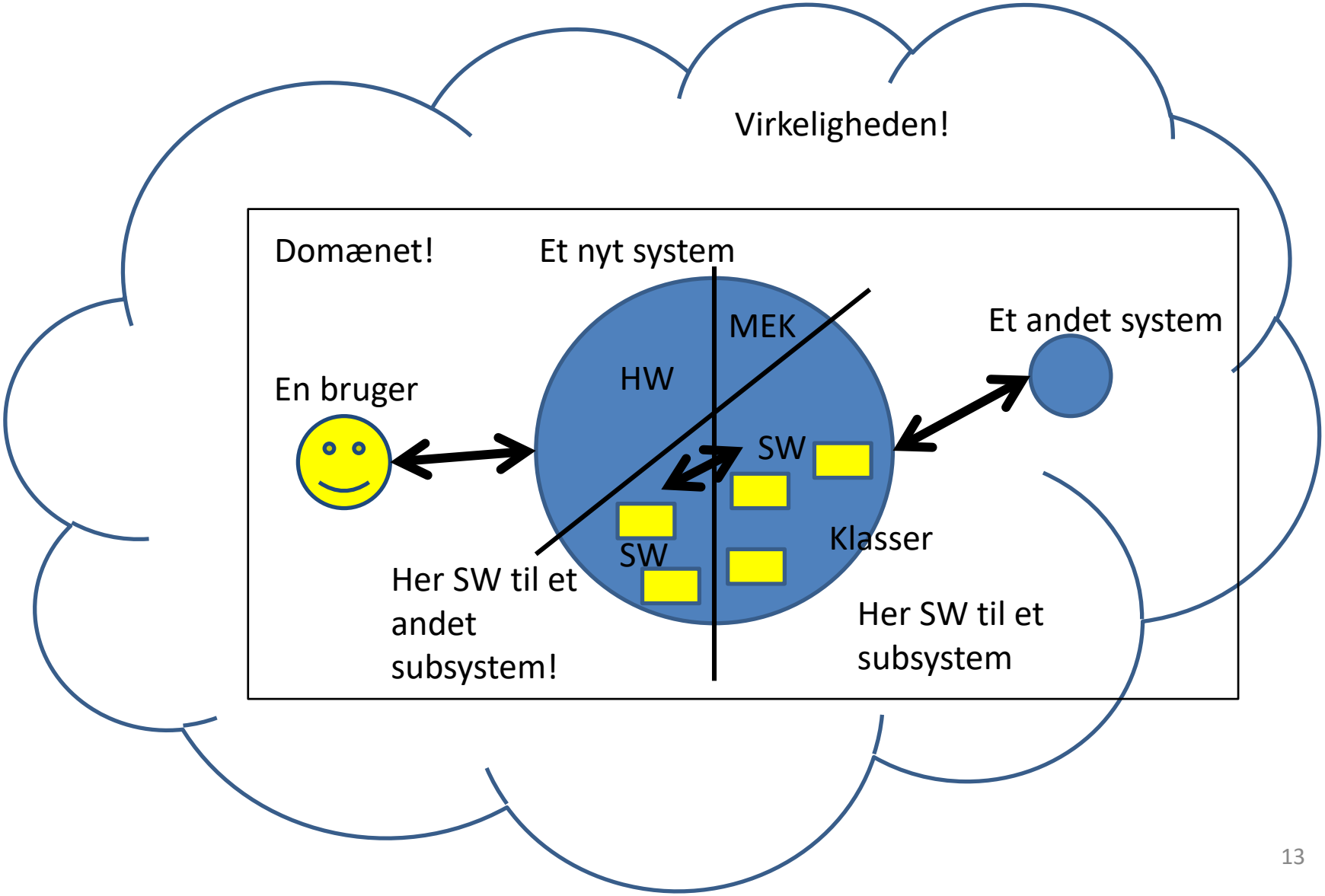
- Man må gerne bruge en UC **Actor** (tændstiksmand) på Applikationsmodellens sekvensdiagram – **MEN:**
  - Den faktiske **Actor** og ***boundary* klassen/r** for aktøren og den ***domain* klasse** som bruges til at gemme attributter for aktøren – er 3 forskellige ting!
  - En faktisk **Actor har IKKE metoder** og **kan IKKE kalde metoder** – det kan kun ***boundary* klassen/r for** aktøren!
    - Ikke alle tegneværktøjer kan tegne messages uden () på sekvensdiagrammer :-(

Guideline Actor



- Løsning Smartfridge

Virkeligheden og systemet – med subsystemer!



Applikationsmodellen – Step 1
Version 3!

- Applikationsmodellen opbygges skridt for skridt, hvor hvert skridt styres af én UC
- **Èn for hvert subsystem!**

Step 1.1:	Vælg den næste fully-dressed UC til at designe for
Step 1.2a:	Identificer alle involverede **aktører og subsystemer** i UC **for dette subsystem** → ***Boundary* klasser**
Step 1.2b:	Hvis man har et IBD som definerer de faktiske hardware interfaces**, også mellem subsystemerne**: opsplit *Boundary* klasserne i relevante *hardware interface Boundary* klasser!
Step 1.3:	Identificer **relevante klasser i Domænemodellen** som er involveret i UC → ***Domain* klasser**
Step 1.4:	Tilføj én UC *control* → ***Control* klasse**

Applikationsmodellen – Step 2
Version 3

- Samarbejdet mellem klasserne udledes nu fra UC **og fra System sekvensdiagrammer for UC**
- **For hvert subsystem!**

Step 2.1: Gennemgå UC's hovedscenarie skridt-for-skridt **og/eller System sekvensdiagrammet for UC** og udtænk hvordan klasserne kan samarbejde for at udføre skridtet!
Man skal kun se på de skridt/messages, der involverer det pågældende subsystem!

Step 2.2: Opdater sekvens- og klassediagrammet for at beskrive samarbejdet (metoder, associationer, attributter)

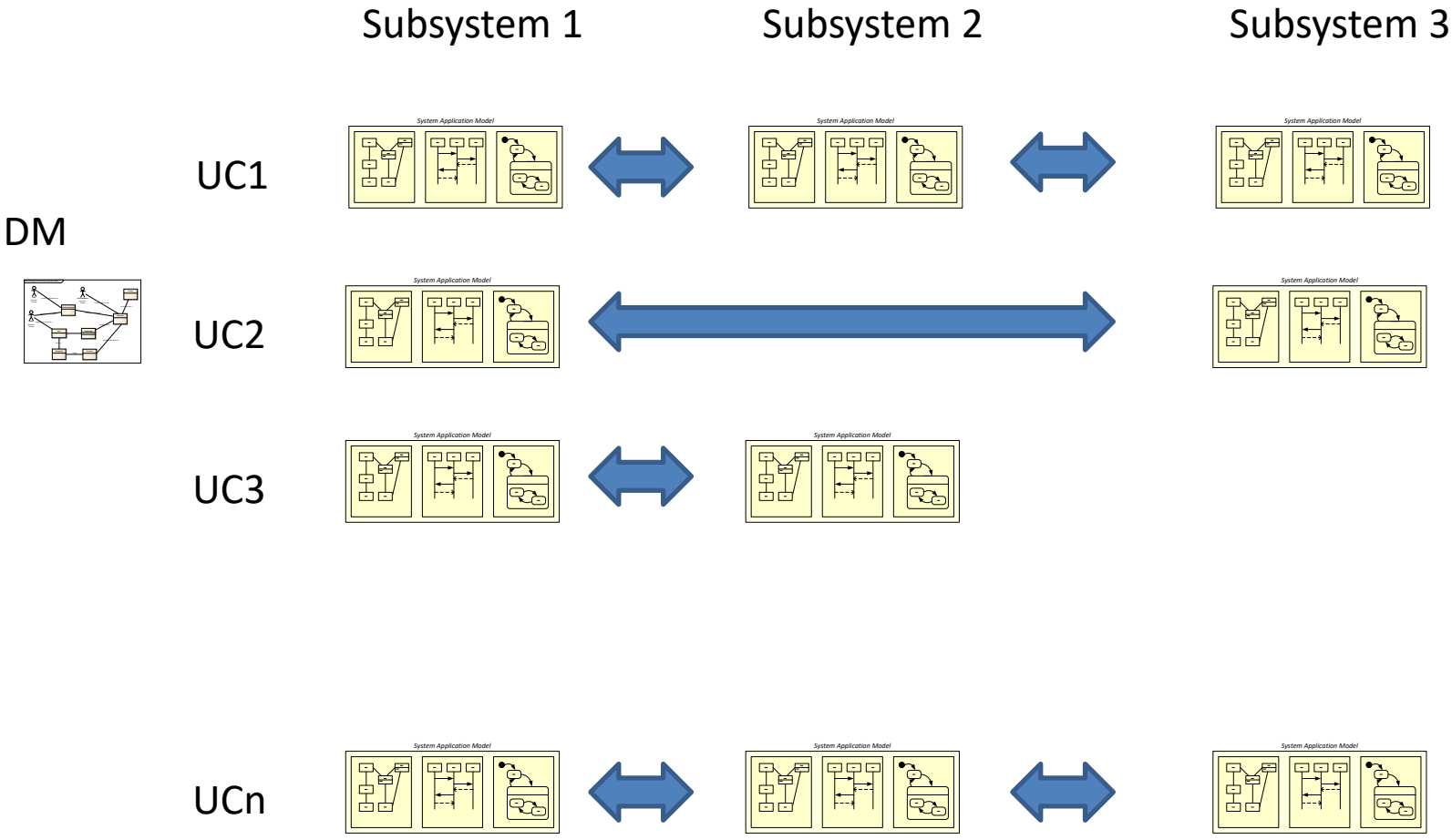
Step 2.3: Hold øje med, om der er state-baserede aktiviteter og opdater STMs for disse klasser (tilstande, triggere, overgange, aktioner)
(Step 2.3 springes over hvis der ikke er nogen tilstandsbaserede klasser)

Step 2.4: Verificer at diagrammerne passer med UC (postconditions, test)

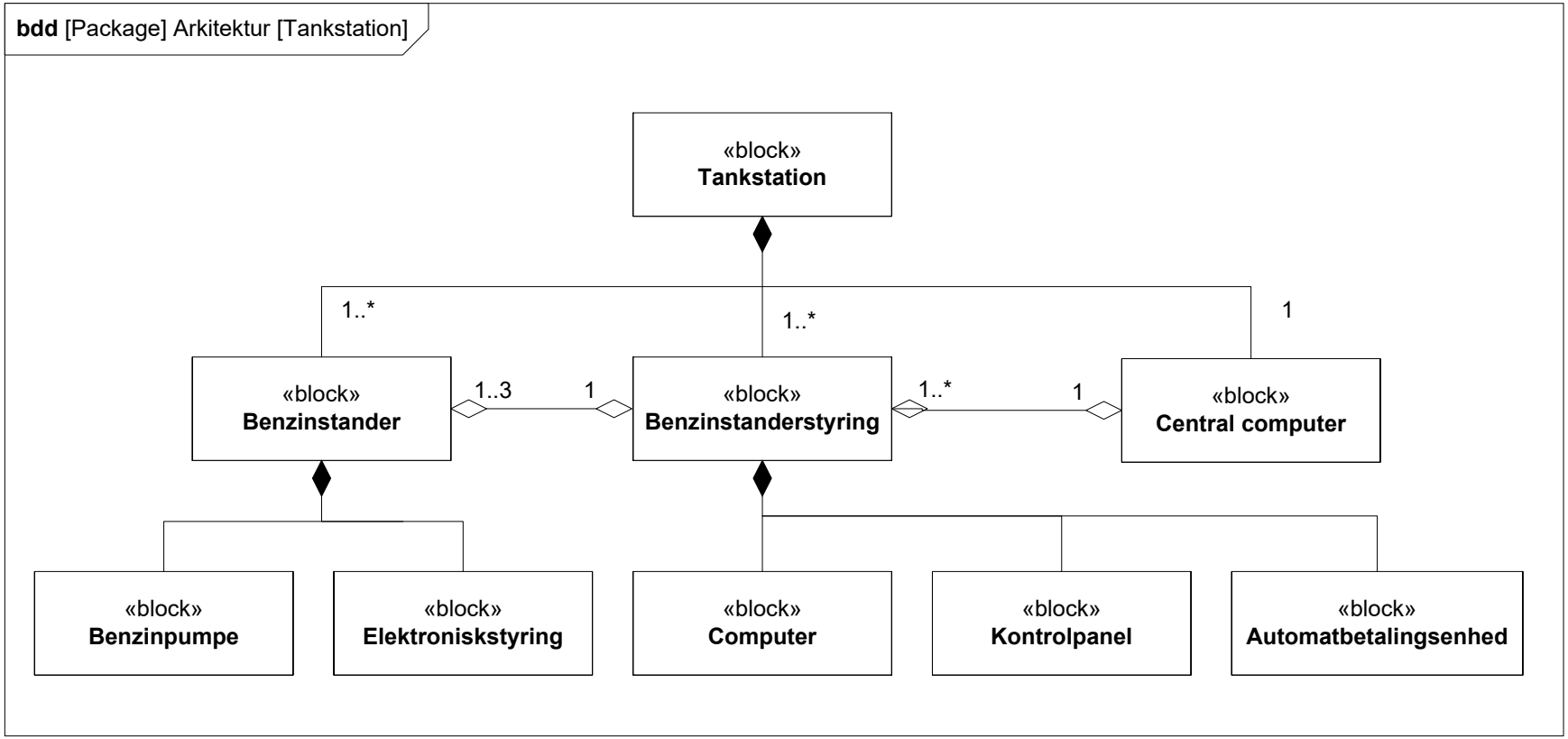
Step 2.5: Gentag 2.1 – 2.4 for alle UC extentions. Finpuds modellen.

- Alle 3 diagrammer (cd, SEQ, STM) opdateres *parallelt/samtidigt* **for et subsystem af gangen** under dette arbejde

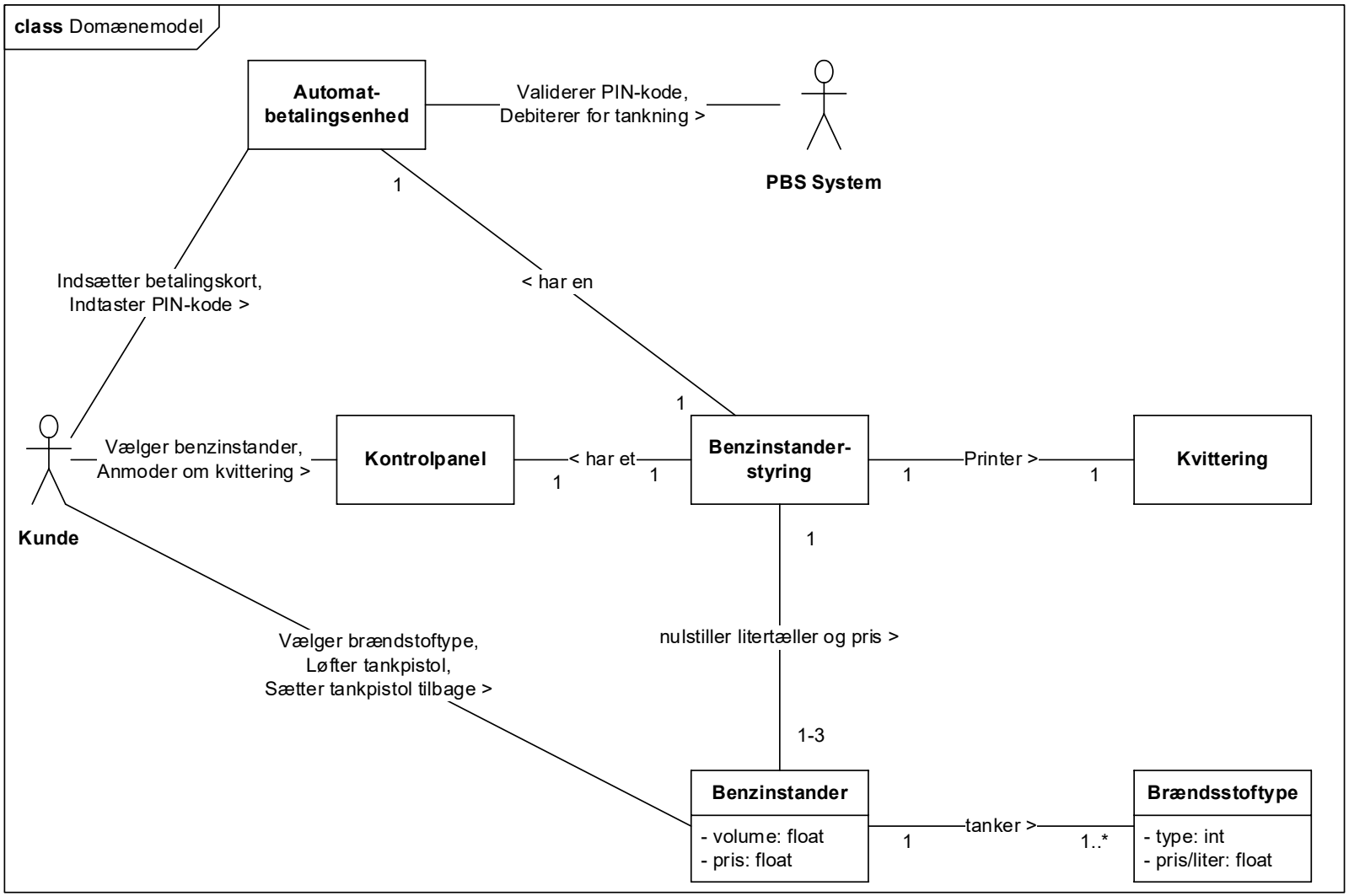
Applikationsmodeller for samarbejdende subsystemer



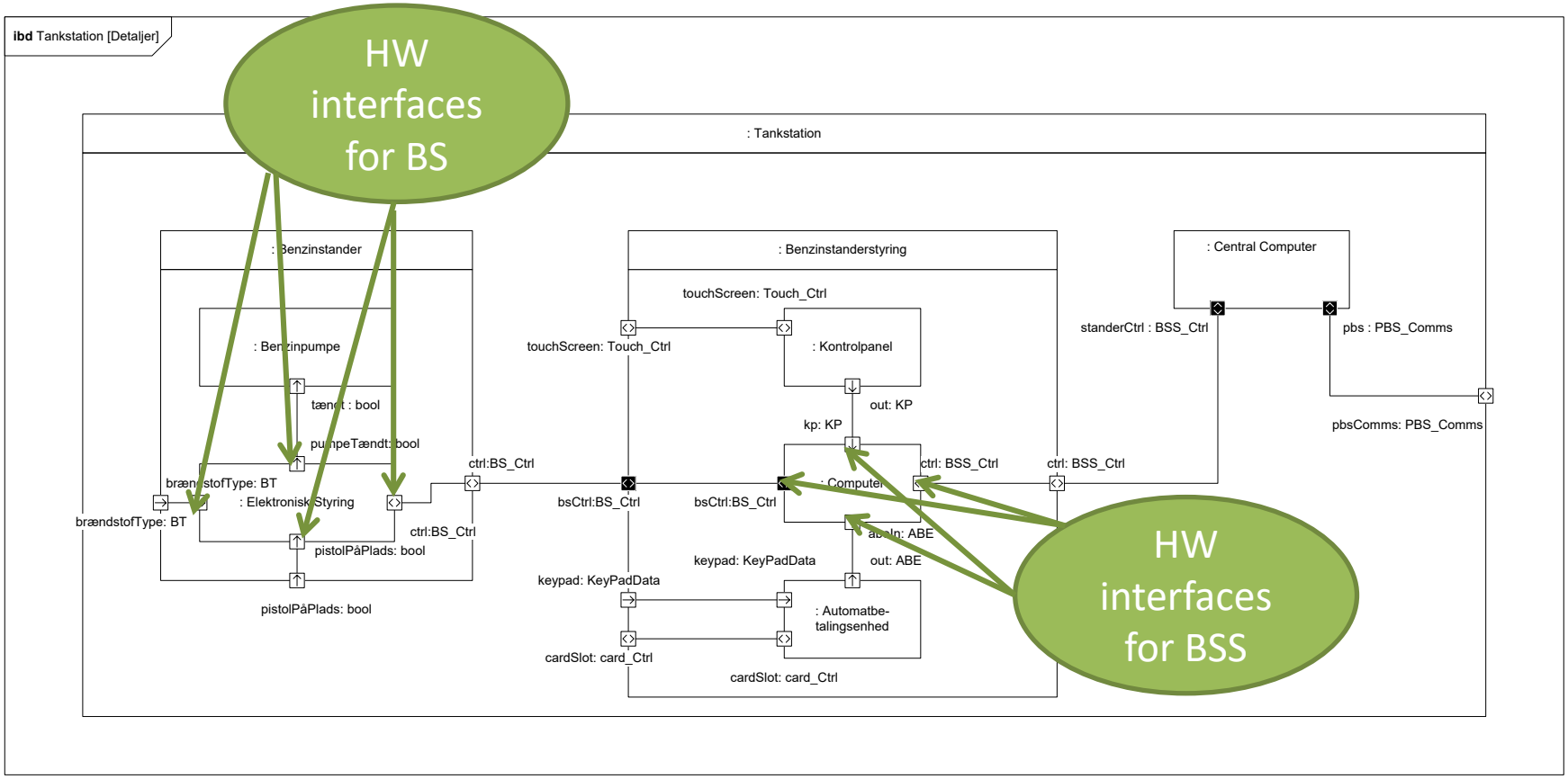
System med subsystemer



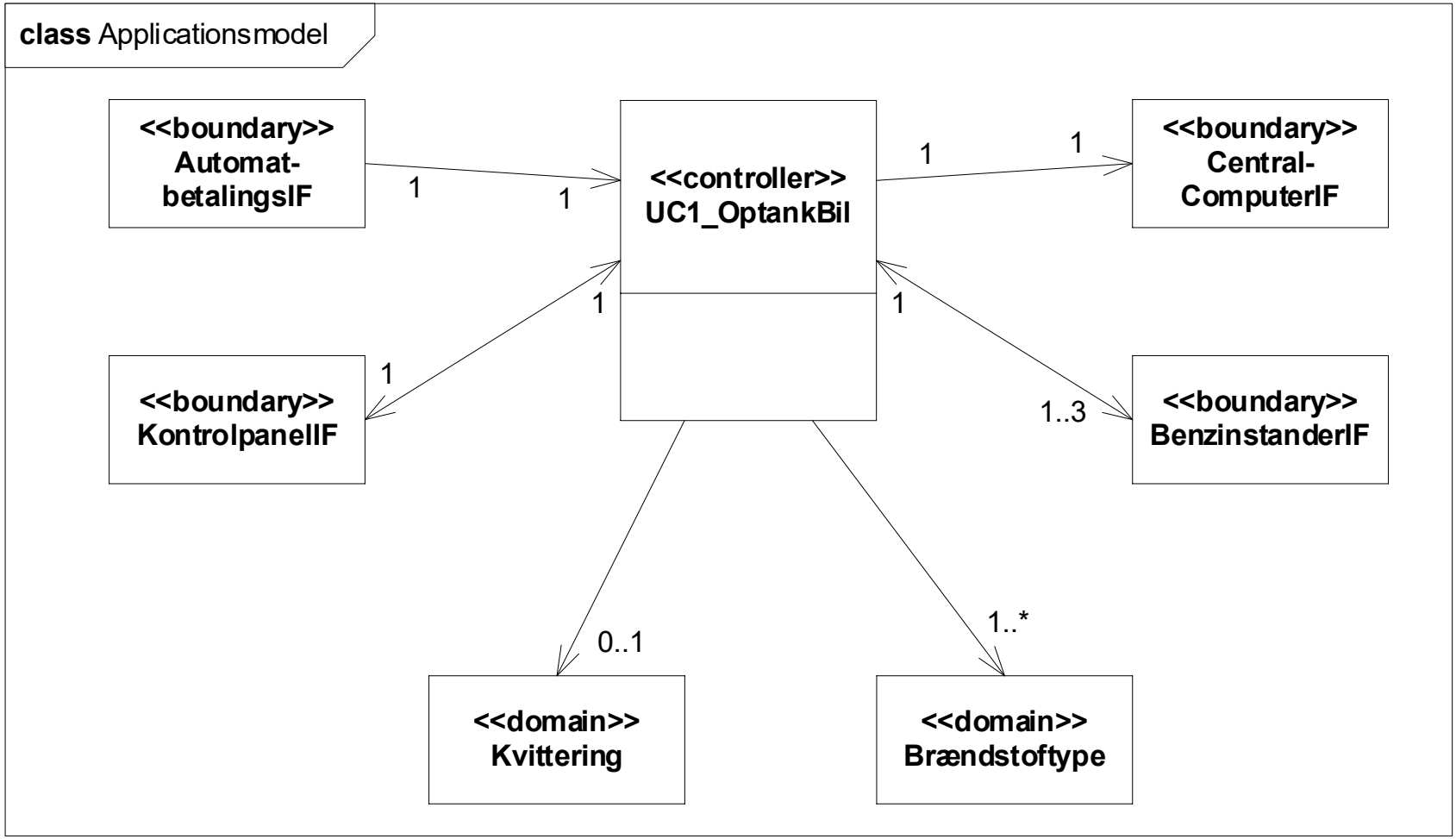
Domænemodellen for hele systemet



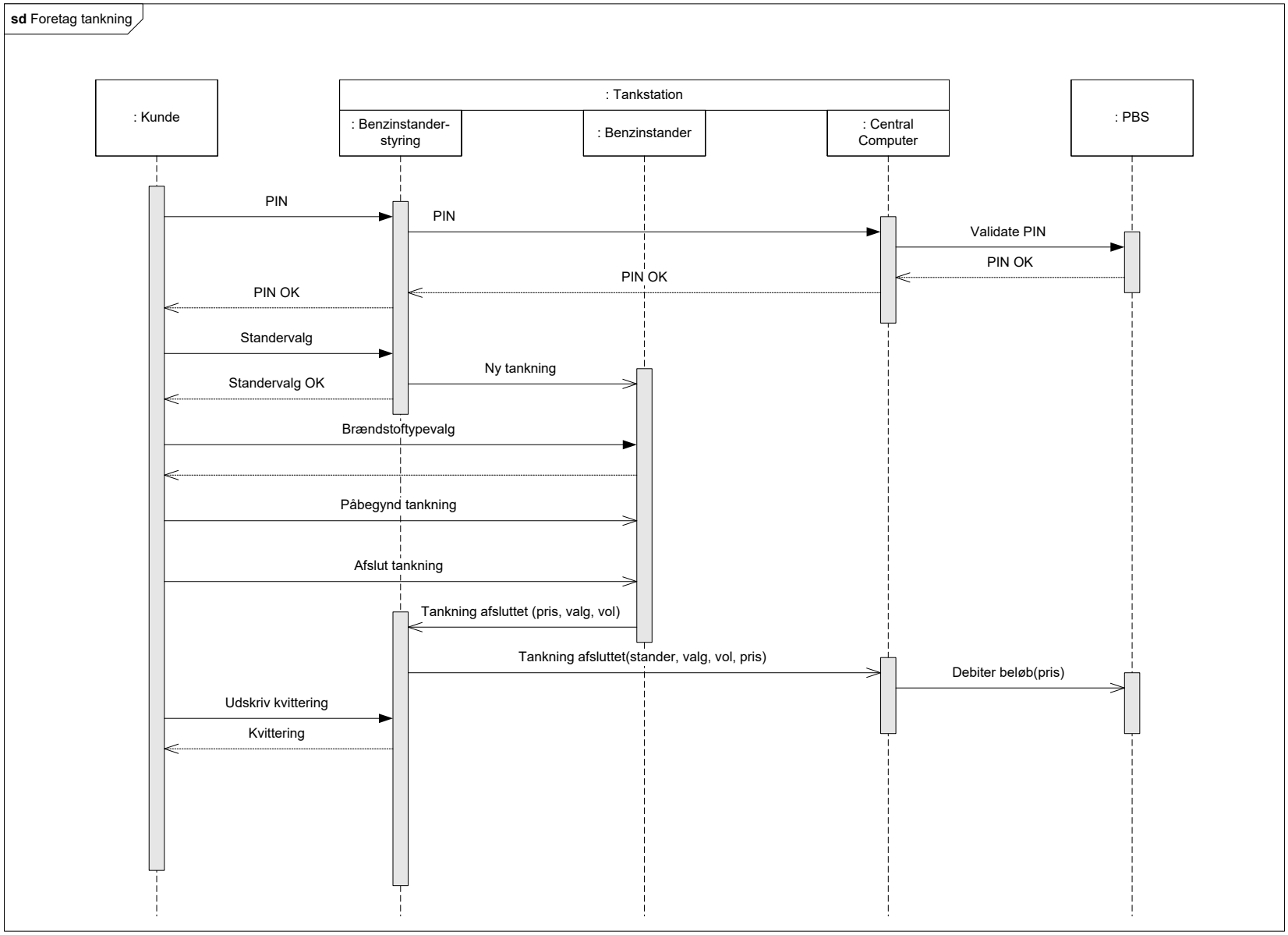
IBD for systemet



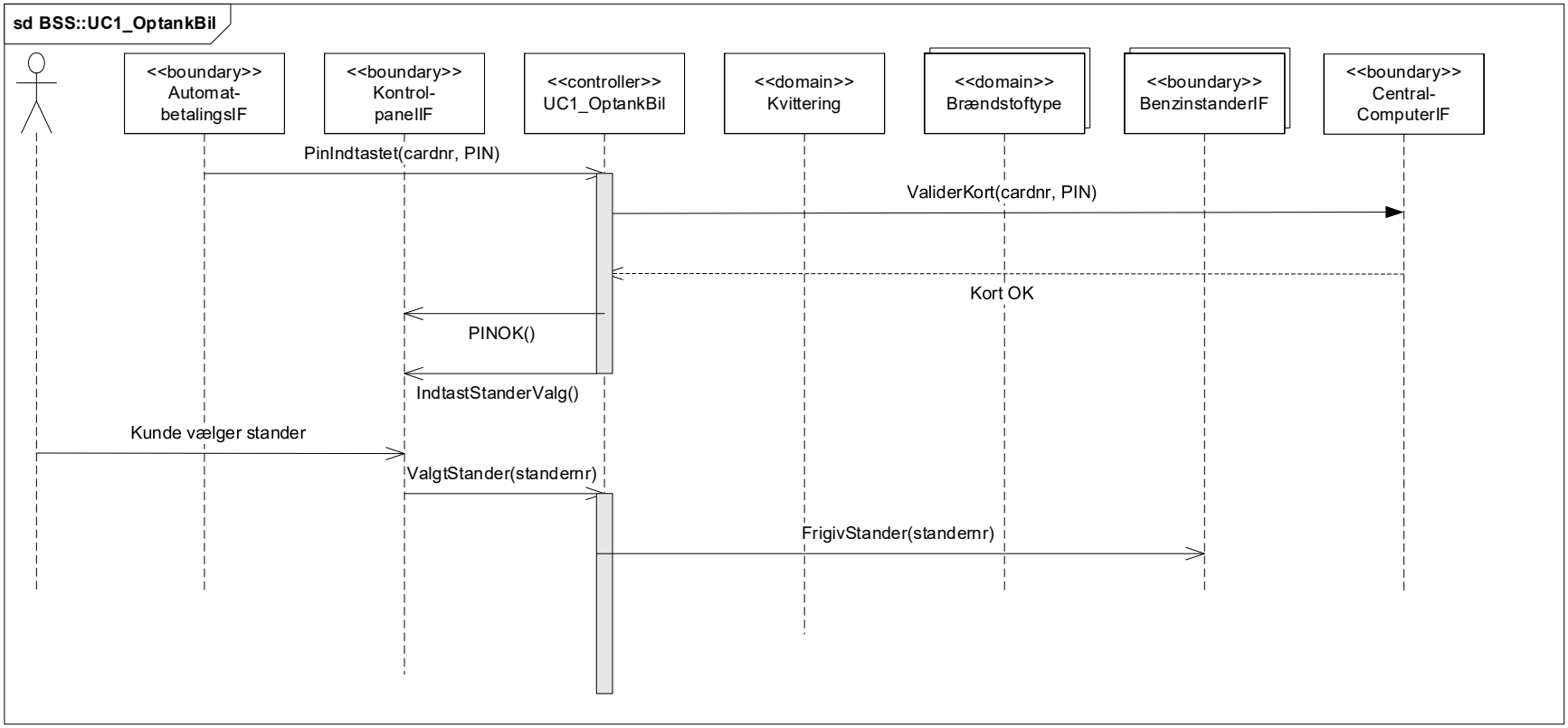
1. Version af klassediagram for Benzinstanderstyringen (BSS)



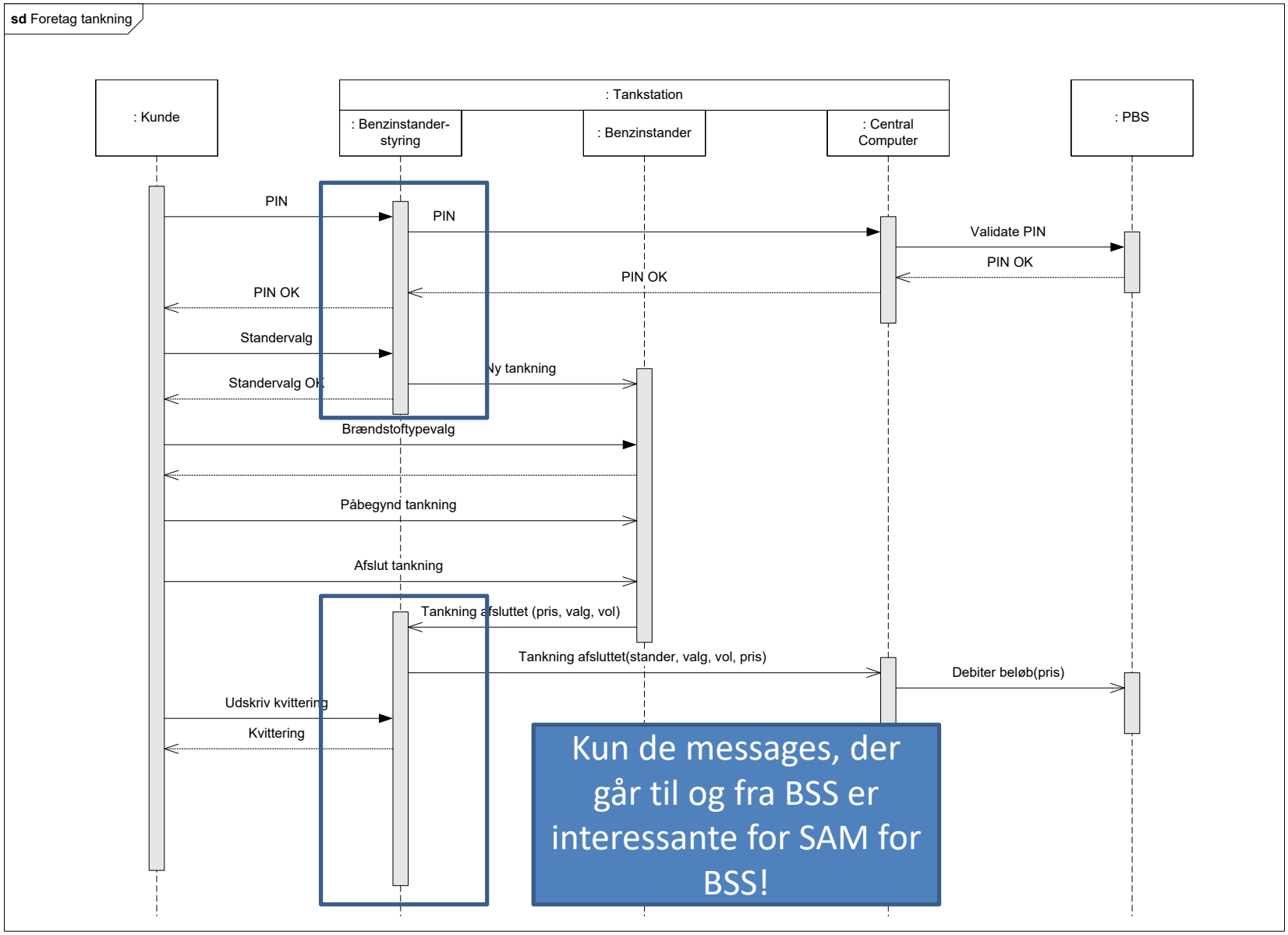
System SD for UC Optank Bil



Start på Applikationsmodellens SD for BSS for UC Optank Bil



System SD for UC Optank Bil



Your turn: System Application Model for Benzinstanderstyring – UC Optank Bil

- Førdiggør **Applikationsmodellen** for **subsystemet Benzinstanderstyringen** for "UC Optank Bil"
- Brug som input
  - BDD og IBD
  - System sekvensdiagrammet
  - Domænemodellen
  - Den 1. version af klassediagrammet (se ovenfor)

1. Check at steps 1.1-1.4 er gennemført korrekt for den 1. version af klassediagrammet
2. Gennemfør steps 2.1-2.5 ved at fortsætte med at arbejde med 1. version af klassediagrammet og lav det tilhørende sekvensdiagram
3. Tænk og check om der mangler domæneklasser, hardware interface klasser, etc, eller om der nogen, der er overflødige **for dette subsystem**!