Wat SysML voorbeelddiagrammen

[3 april 2022, door Marius Versteegen]

De voorbeelden in Delligatti, de reader en de sheets kunnen eigenlijk niet gebruikt worden zonder een verhaal erbij van wat er anders of beter aan moet in het architectuurdocument.

Ik heb mijn best gedaan om er een goed voorbeeld bij te maken. Ik heb nog geen tijd gehad om het zelf goed na te kijken – er zitten dus ongetwijfeld nog fouten in – en ook de begeleidende teksten zijn nog niet compleet – en ben ik niet de diepte in gegaan – maar het geeft verder hopelijk een aardig beeld van wat ik verwacht van sommige diagrammen.

Als kapstok heb ik gekozen voor een kattenvoermachine. Het moet het mogelijk maken om meerdere katten zonder hoofdzorgen en soeza automatisch een persoonlijk getailored dieet te laten volgen. Bijvoorbeeld rode katten worden sneller dik dan zwarte, en hebben een strakker dieet nodig.

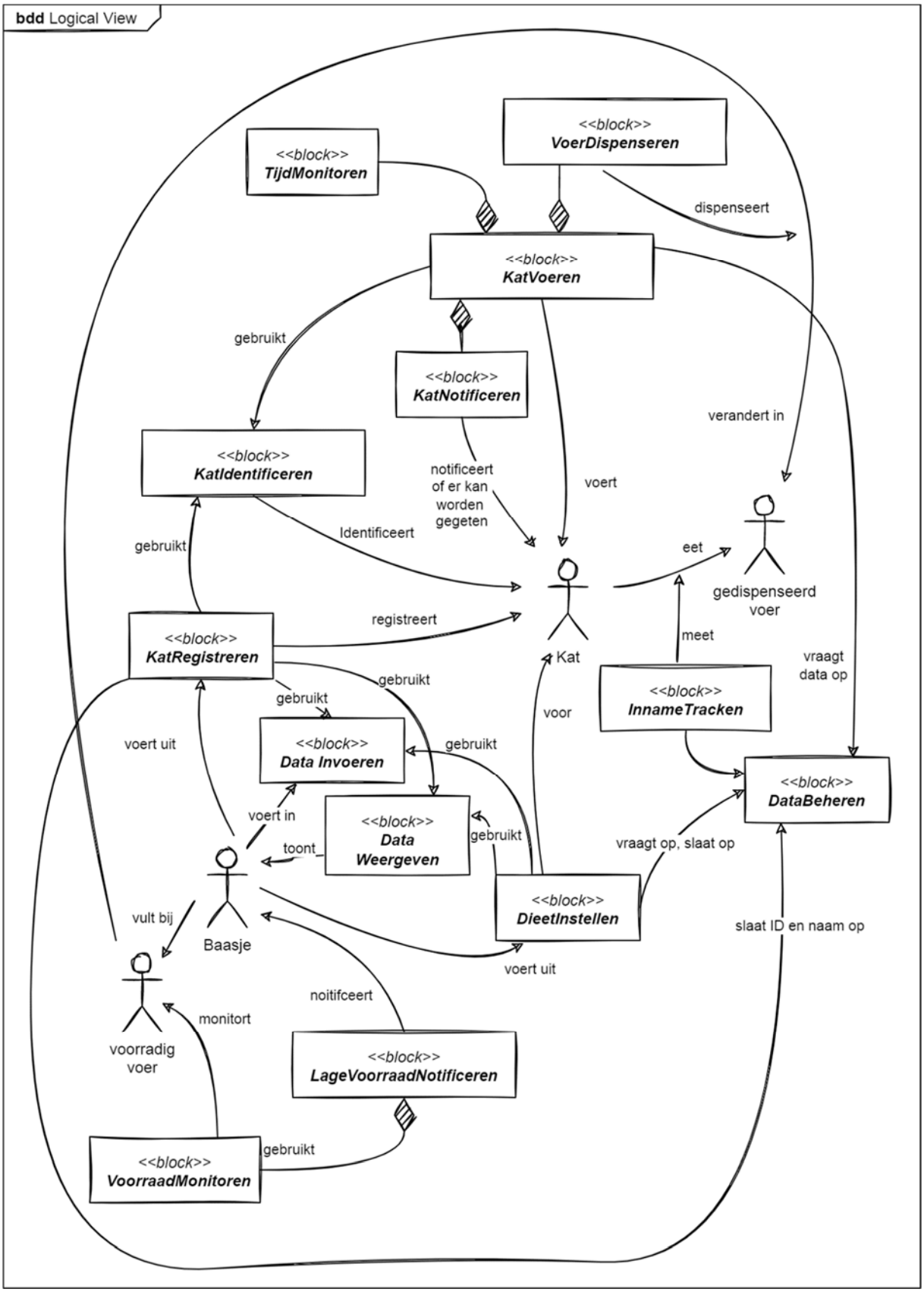
Ik heb nog geen tijd gevonden voor het uitschrijven van usecases en requirements. In plaats daarvan laat ik de CAF voor nu bij onderstaande samenvatting.

Het idee is globaal: een kattenvoermachine met een centrale voeropslag en meerdere “tunnels”: voor elke kat een aparte tunnel (zodat katten kunnen leren wat hun tunnel is en in harmonie gelijktijdig kunnen eten). In elke tunnel heeft elke kat een eigen voerbak. Aan een rustig lichtjespatroon oid boven zijn tunnel, kan de kat zien of er nog eet-quotum voor hem is. Zodra een kat de tunnel in loopt, herkent het systeem de kat (aan de chip in zijn nek). Als het de kat betreft die hoort bij de betreffende tunnel en de kat nog voedingsquotum heeft voor het betreffende dagdeel, zorgt het systeem dat er wat brokjes in zijn bakje worden gedeponeerd. Als de kat de brokjes heeft opgegeten, worden er opnieuw brokjes gedeponeerd. Dat herhaalt zich totdat de kat wegloopt of dat hij zijn dagdeel-quotum op heeft. De tijdstippen van de dagdelen en de bijbehorende quota kunnen per kat worden ingevuld via een webinterface (dus vanaf elke browser). Eenmalig moeten eerst de chipIDs van de katten geregistreerd worden: even de kat een tunnel inlokken (even een dispense-portie genereren via een knopje in de web-interface) en de naam dan de kat linken aan zijn chipID. Als de voer-voorraad onder een bepaald peil komt, wordt het baasje genotificeerd (bijvoorbeeld met een duidelijk lichteffect).

Logische view

De logische view geeft een functionele decompositie weer. Het geeft weer welke functies het systeem allemaal aanbiedt om de usecases te kunnen waarmaken en hoe ze met elkaar in verband staan. Daarbij wordt zoveel mogelijk voorkomen om daarbij (al) beslissingen te nemen over de hardware waarin het zal worden gerealiseerd. Die beslissingen komen in een latere fase wel, bijvoorbeeld na gebruikmaking van beslissingstabellen. Door ook de actoren op te nemen wordt duidelijk gemaakt welke functionaliteit in dienst staat van welke actors.

De functies in deze logische view zijn als het goed is ook in enige vorm te vinden in de functionele requirements. De samenhang van de functies onderling en met de actoren is als het goed is terug te vinden in de usecases.


bdd Logical View
<<block>>
VoerDispenseren
<<block>>
TijdMonitoren
dispenseert
<<block>>
KatVoeren
gebruikt
<<block>>
KatNotificeren
verandert in
<<block>>
KatIdentificeren
notificeert
of er kan
worden
gegeten
voert
Identificeert
gebruikt
eet
gedispenseerd
voer
registreert
<<block>>
KatRegistreren
Kat
meet
vraagt
data op
gebruikt
gebruikt
<<block>>
InnameTracken
voor
voert uit
<<block>>
Data Invoeren
gebruikt
<<block>>
DataBeheren
voert in
<<block>>
Data
Weergeven
gebruikt
toont
vraagt op, slaat op
<<block>>
DieetInstellen
vult bij
Baasje
slaat ID en naam op
voert uit
noitifceert
monitort
voorradig
voer
<<block>>
LageVoorraadNotificeren
gebruikt
<<block>>
VoorraadMonitoren

Software BDD

De software BDD geeft een software architectuur weer van de kattenvoermachine. Voor ons architectuurdocument beperken we ons tot de software van onze “hoofdcomputer”. Om een en ander goed leesbaar te houden op A4 is de Software BDD opgesplitst in 3 verschillende BDDs:

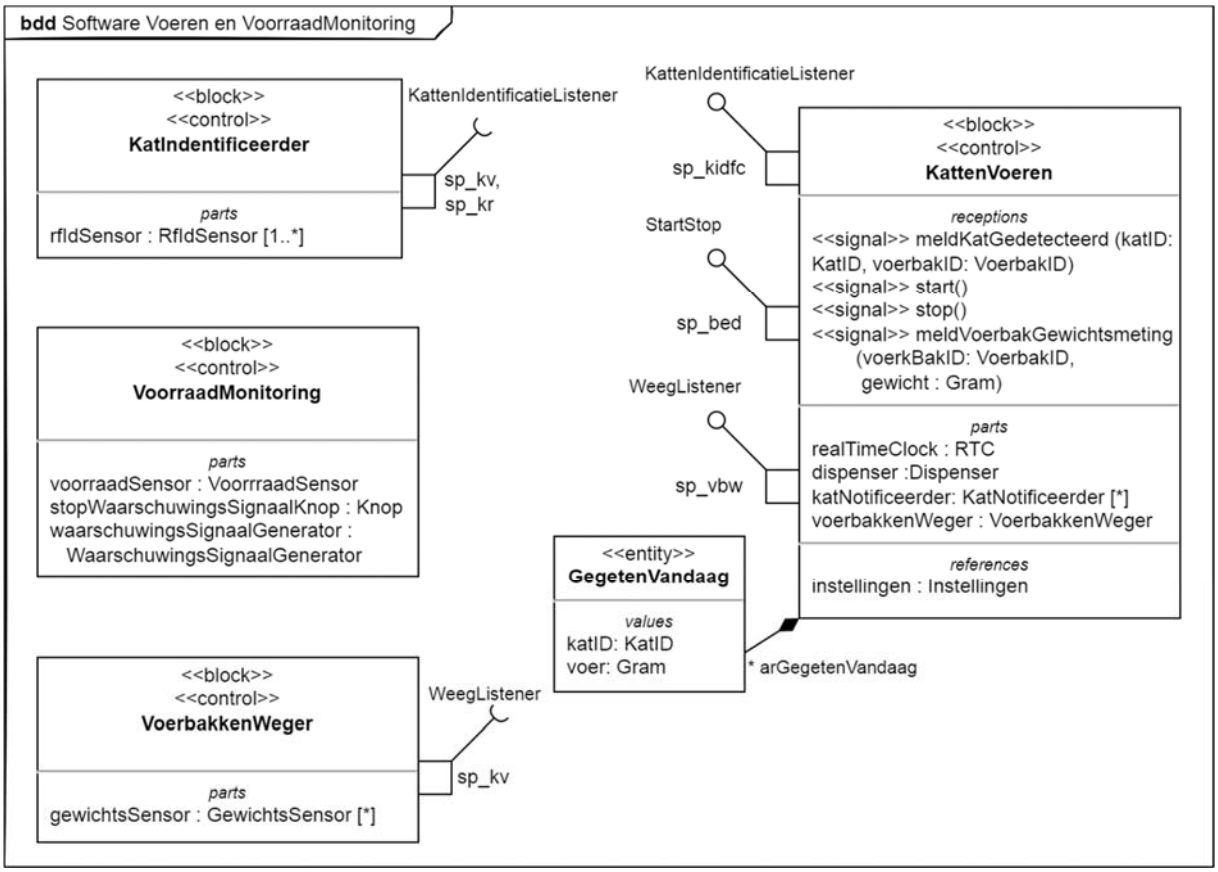
- BDD Software Voeren en VoorraadMonitoring
- BDD Software Instellen
- BDD Software Interfaces en ValueTypes

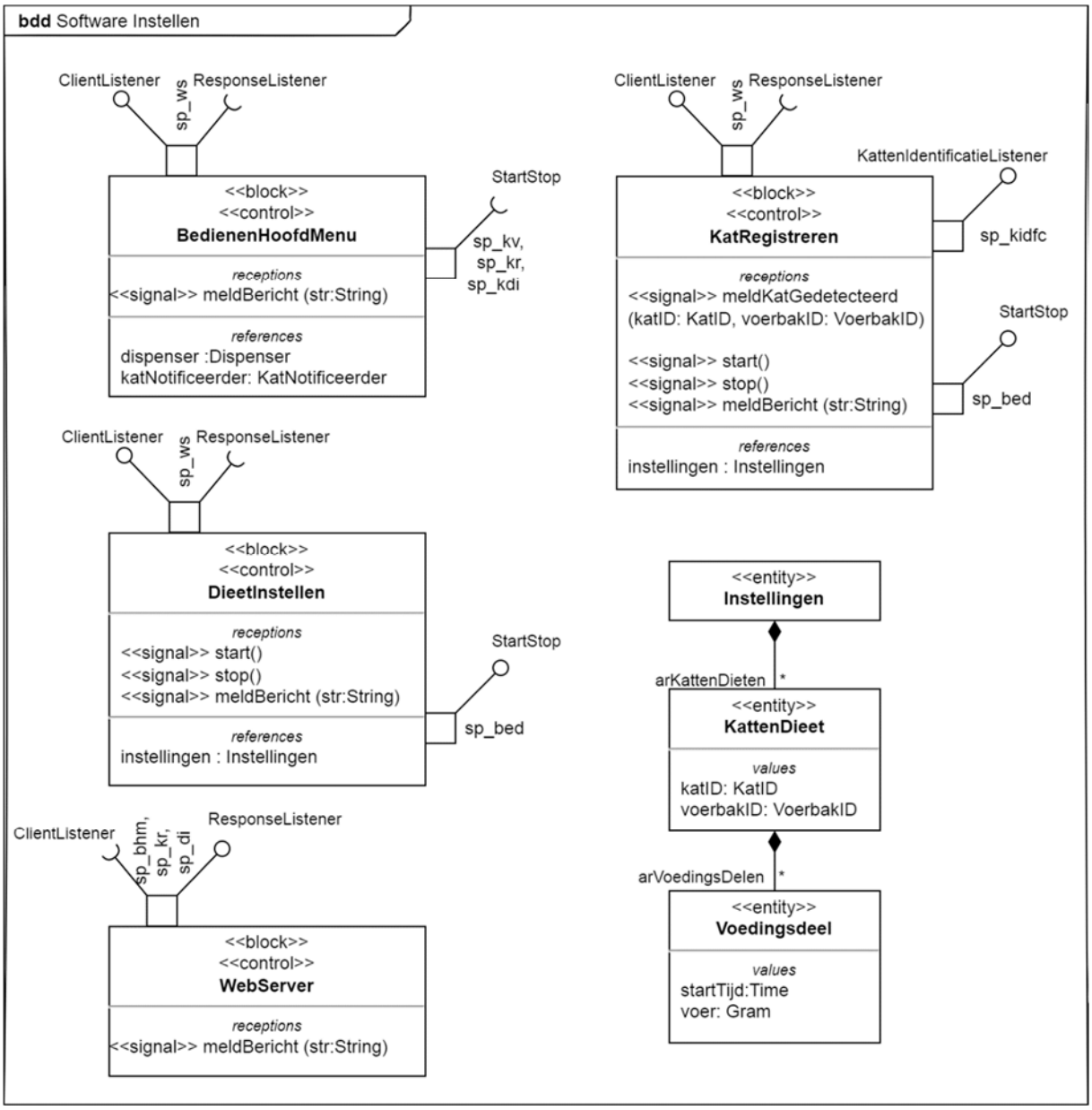
Een architectuurdocument is geen gedetailleerde uitwerking (zoals wat system engineers maken), maar een zo duidelijk mogelijk overzicht van de belangrijkste zaken.

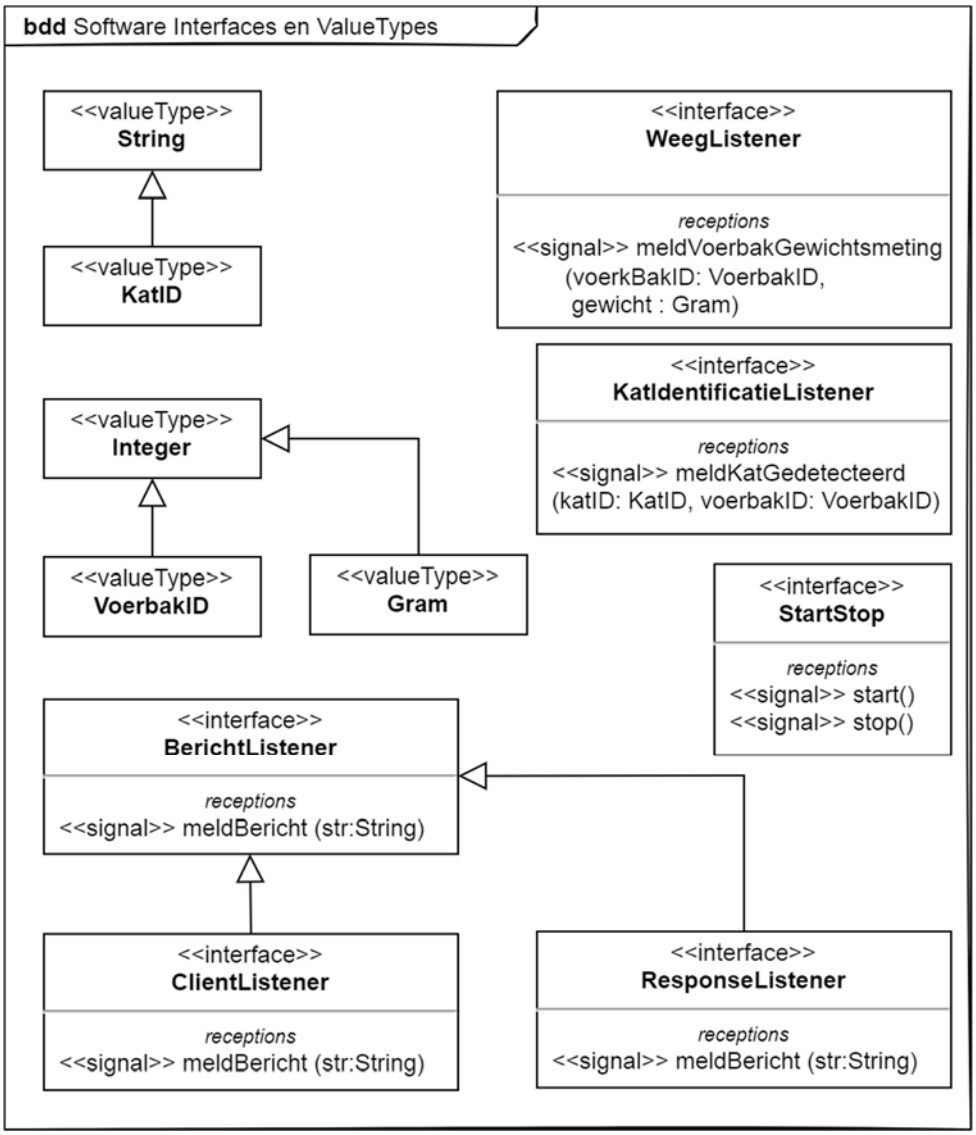
Met die gedachtengang in het achterhoofd heb ik er hier voor gekozen om trivialiteiten zoals het registreren van listeners en het gebruik van setters en getters, weg te laten.

De ClientListener en ResponseListener zijn (op dit moment nog) identiek. Toch heb ik ervoor gekozen om er aparte klassen van te maken zodat de rollen van de klassen die ervan afleiden duidelijk worden. Deze twee interfaces vormen de back-and-forth communicatie met de WebServer klasse. Vandaar dat ze zijn samengevoegd in de standard ports.

BedienenHoofdMenu is de klasse die de hoofdverantwoordelijkheid heeft voor de bediening. De interface van die bediening loopt via de WebServer klasse. Het baasje kan via een browser met die klasse data uitwisselen. Gedurende het instelproces zorgt BedienenHoofdMenu via de StartStop interfaces ervoor dat het voeren tijdelijk wordt stilgelegd en dat de juiste submenus (KatRegistreren en DieetInstellen) indien nodig worden geactiveerd.

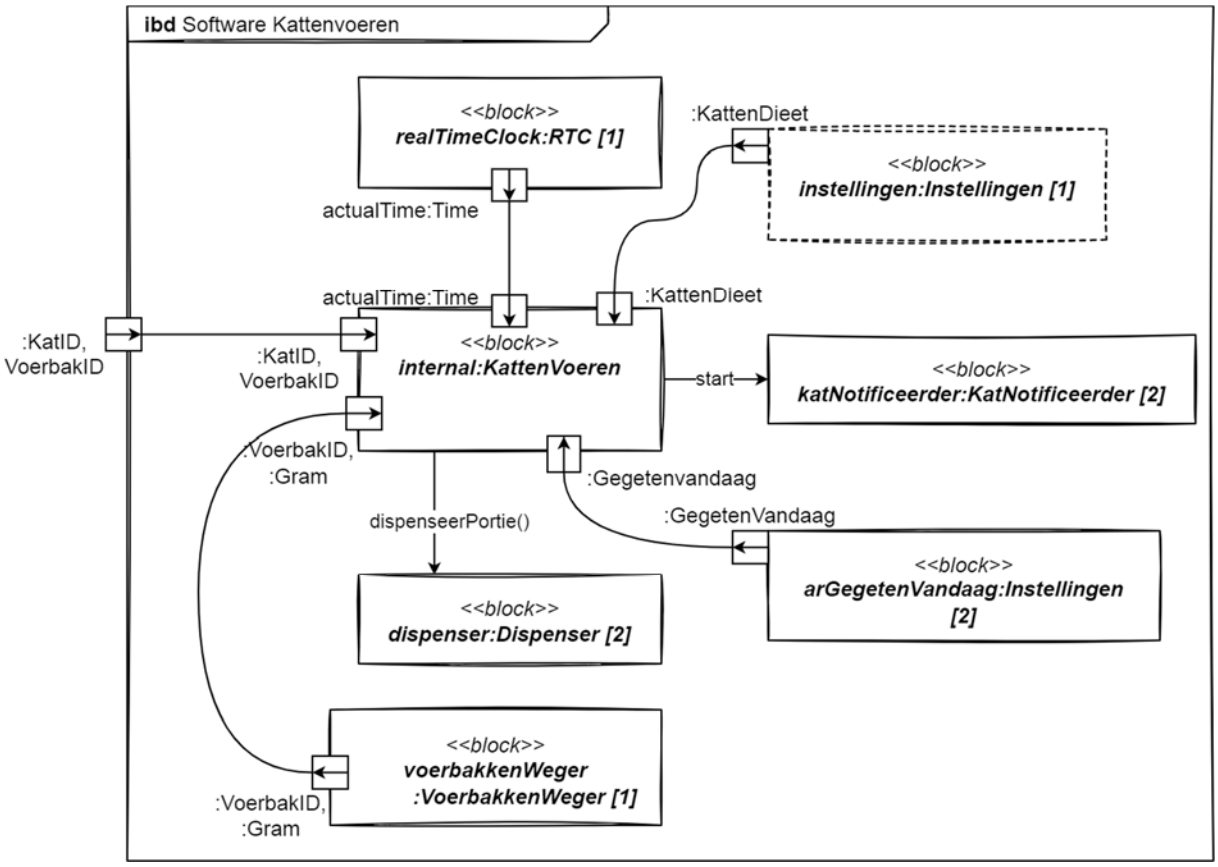

bdd Software Voeren en VoorraadMonitoring
<<block>>
<<control>>
KatIndentificeerder
parts
rfIdSensor : RfIdSensor [1..*]
KattenIdentificatieListener
sp_kv,
sp_kr
KattenIdentificatieListener
sp_kidfc
<<block>>
<<control>>
KattenVoeren
receptions
<<signal>> meldKatGedetecteerd (katID: KatID, voerbakID: VoerbakID)
<<signal>> start()
<<signal>> stop()
<<signal>> meldVoerbakGewichtsmeting (voerkBakID: VoerbakID, gewicht : Gram)
StartStop
sp_bed
WeegListener
sp_vbw
parts
realTimeClock : RTC
dispenser :Dispenser
katNotificeerder: KatNotificeerder [*]
voerbakkenWeger : VoerbakkenWeger
references
instellingen : Instellingen
<<block>>
<<control>>
VoorraadMonitoring
parts
voorraadSensor : VoorrraadSensor
stopWaarschuwingsSignaalKnop : Knop
waarschuwingsSignaalGenerator : WaarschuwingsSignaalGenerator
<<entity>>
GegetenVandaag
values
katID: KatID
voer: Gram
* arGegetenVandaag
<<block>>
<<control>>
VoerbakkenWeger
parts
gewichtsSensor : GewichtsSensor [*]
WeegListener
sp_kv


bdd Software Instellen
ClientListener
sp_ws
ResponseListener
<<block>>
<<control>>
BedienenHoofdMenu
receptions
<<signal>> meldBericht (str:String)
references
dispenser :Dispenser
katNotificeerder: KatNotificeerder
StartStop
sp_kv,
sp_kr,
sp_kdi
ClientListener
sp_ws
ResponseListener
KattenIdentificatieListener
<<block>>
<<control>>
KatRegistreren
receptions
<<signal>> meldKatGedetecteerd
(katID: KatID, voerbakID: VoerbakID)
<<signal>> start()
<<signal>> stop()
<<signal>> meldBericht (str:String)
references
instellingen : Instellingen
sp_kidfc
StartStop
sp_bed
ClientListener
sp_ws
ResponseListener
<<block>>
<<control>>
DieetInstellen
receptions
<<signal>> start()
<<signal>> stop()
<<signal>> meldBericht (str:String)
references
instellingen : Instellingen
StartStop
sp_bed
<<entity>>
Instellingen
arKattenDieten
*
<<entity>>
KattenDieet
values
katID: KatID
voerbakID: VoerbakID
arVoedingsDelen
*
<<entity>>
Voedingsdeel
values
startTijd:Time
voer: Gram
ClientListener
sp_bhm,
sp_kr,
sp_di
ResponseListener
<<block>>
<<control>>
WebServer
receptions
<<signal>> meldBericht (str:String)



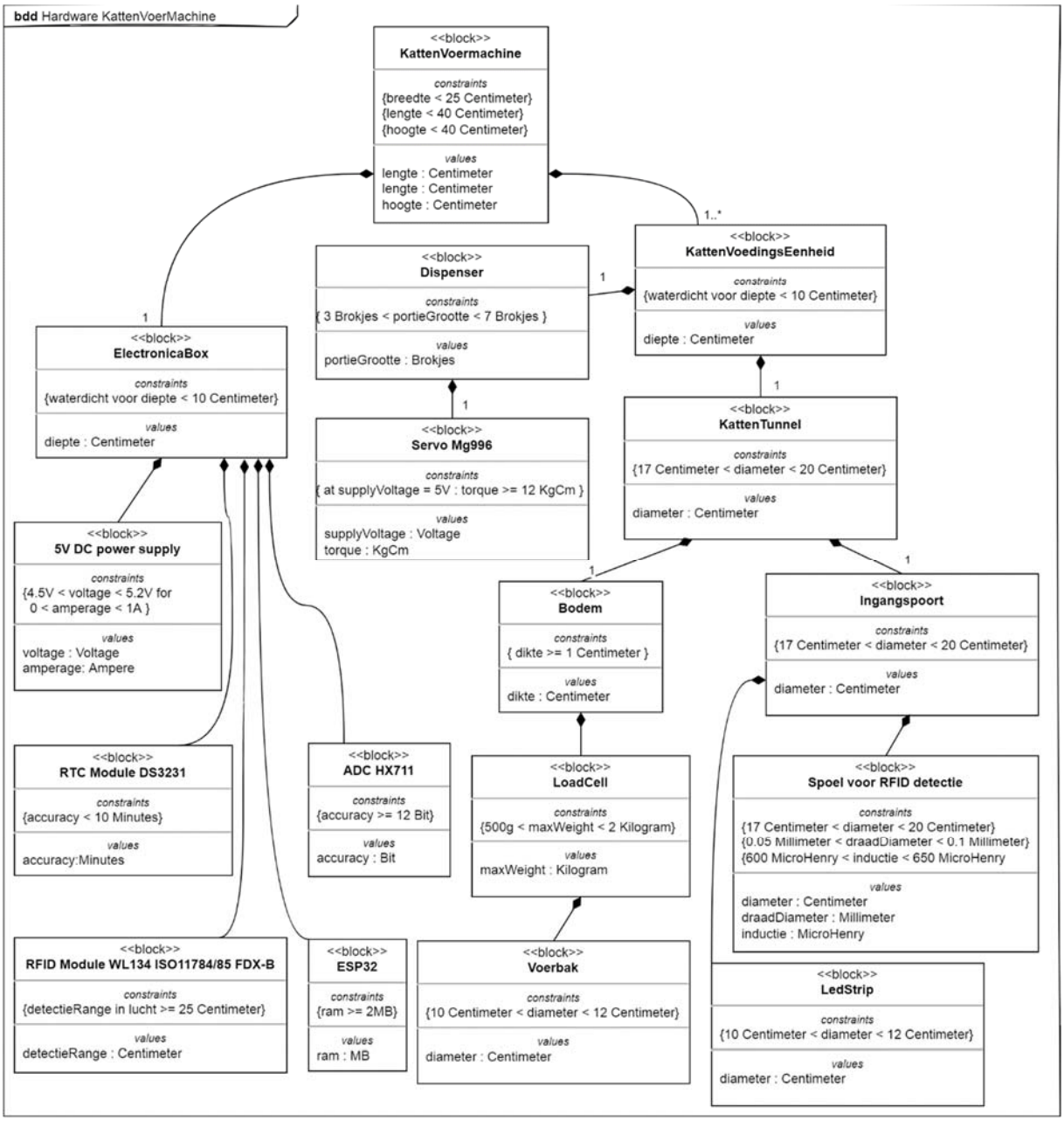
Software IBD

Onderstaand is een IBD te zien van de klasse KattenVoeren. Het heeft veel weg van een activity-diagram. De stromen zijn nu echter niet tussen activities maar tussen instanties (geinstanieerde objecten van klassen). De flow ports geven de flow van verschillende typen data weer tussen de parts en references van de instantie van de klasse KattenVoeren. Laatstgenoemde instantie is zelf weergegeven met een object met de naam internal:KattenVoeren. De instantie instellingen:Instellingen is gestippeld, omdat de klasse Kattenvoeren er een reference naar heeft (het is dus geen part, zoals de overige instances). Op de grens van het diagram is er nog een binnenkomende flowport weergegeven. Daar komen de kat detectiegegevens binnen, afkomstig van “elders”. In dit geval is dat een instantie van de KatIdentificeerder. Maar het had ook iets anders kunnen zijn, omdat de klasse KattenVoer er geen part- of reference relatie mee heeft. In deze IBD heb ik voor een realisatie gekozen met 2 katten, die gevoerd worden in ieder een eigen voerbak met een eigen dispenser. Soms is er alleen sprake van een bericht zonder data als payload, zoals dispenseerPortie(). In dat geval lijkt het meer op een “control-flow” en wordt de flow-port weggelaten.



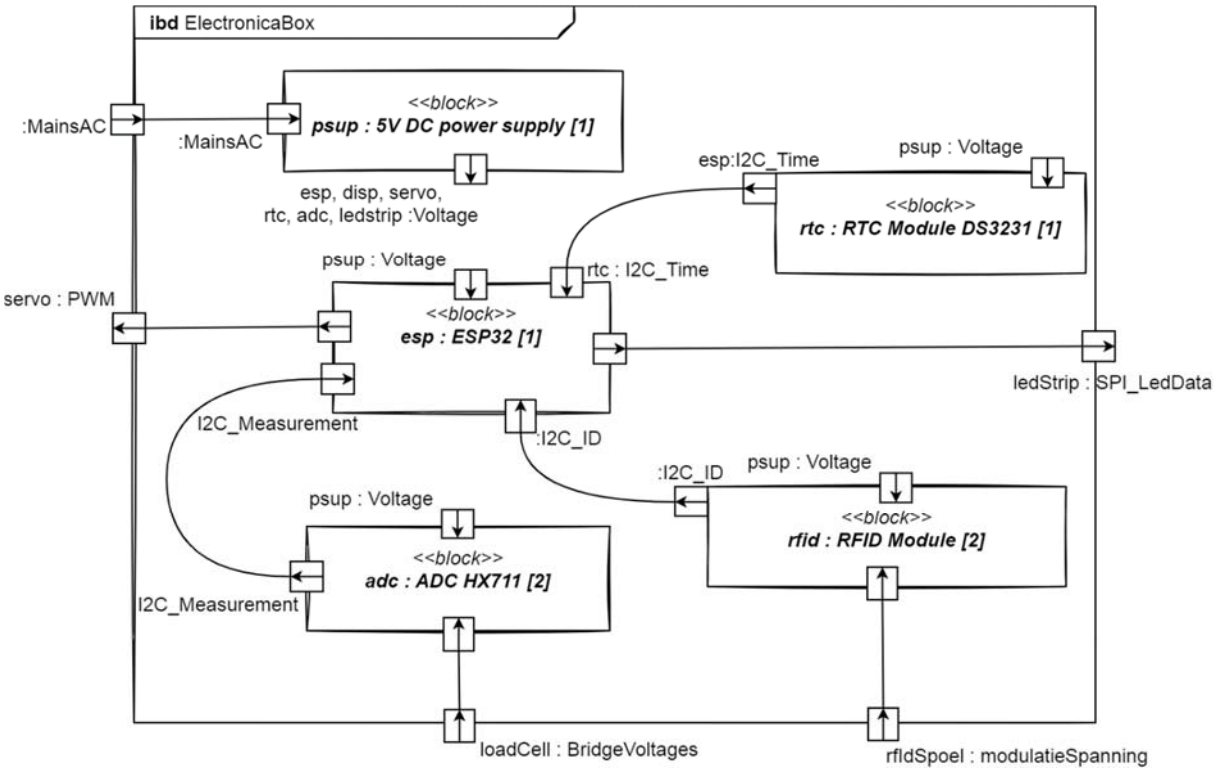
Hardware BDD

Onderstaand diagram toont een fysieke decompositie van de KattenVoermachine. Ik ben nog niet toegekomen aan het toevoegen van alle multiplicities en evt membernamen. Deze BDD lijkt me als uitdrukking van fysieke decompositie al complex genoeg zo. Het uitdrukken van samenhang tussen de electronische modules via flow ports laat ik hier daarom achterwege. Dat komt wel in de IBD.



Hardware IBD

De IBD van de ElectronicaBox laat zien hoe de contents van de ElectronicaBox onderling met elkaar en met de buitenwereld informatie uitwisselen. De box heeft 5 poorten naar de buitenwereld: een voor de stroomvoorziening, een naar de RFID-spoelen, een naar de ledstrips een naar de servos voor de dispensing en een naar de loadcells voor de voerbak-gewichtsmeting.


ibd ElectronicaBox
:MainsAC
:MainsAC
<<block>>
psup : 5V DC power supply [1]
esp, disp, servo,
rtc, adc, ledstrip :Voltage
esp:I2C_Time
psup : Voltage
<<block>>
rtc : RTC Module DS3231 [1]
psup : Voltage
rtc : I2C_Time
servo : PWM
<<block>>
esp : ESP32 [1]
ledStrip : SPI_LedData
I2C_Measurement
:I2C_ID
:I2C_ID
psup : Voltage
psup : Voltage
<<block>>
rfid : RFID Module [2]
<<block>>
adc : ADC HX711 [2]
I2C_Measurement
loadCell : BridgeVoltages
rfIdSpoel : modulatieSpanning