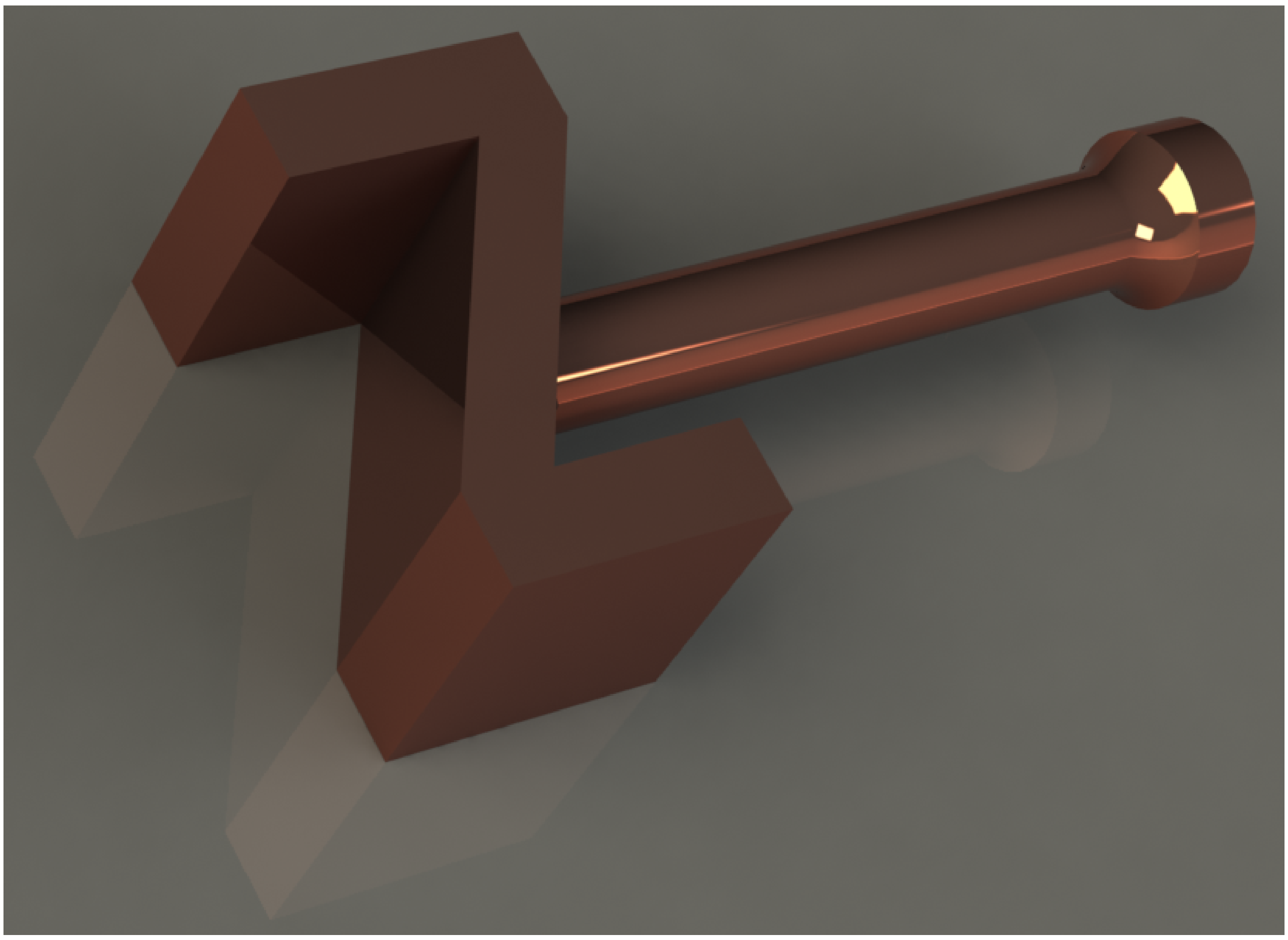
Formal specification - Z spec assignment



Pancake - IoT framework

Anton Krug - 20062210

Contents

1 Introduction **3**

2 Global types and variables **4**

3 Pancake Platform **5**
3.1 PancakePlatform schema 5
3.2 Operations 6
3.2.1 InitPancakePlatform 6
3.2.2 AddLocation 6
3.2.3 RemoveLocation 7
3.2.4 AddThing 8
3.2.5 AddActuatorThing 9
3.2.6 AddData 10
3.2.7 RemoveThing 11
3.2.8 ChangeThingRange 12
3.2.9 ChangeActuatorThingRange 13
3.2.10 FindWhen 13
3.2.11 AddEdge 14
3.2.12 RemoveEdge 15
3.2.13 GetThingStats 15
3.2.14 isThingOver 16
3.2.15 isThingUnder 16

4 PancakeService **17**
4.1 Schemas 17
4.1.1 Customer 17
4.1.2 PancakeService 17
4.2 Operations 18
4.2.1 InitPancakeService 18
4.2.2 RegisterCustomer 18
4.2.3 RegisteredCustomers 18
4.2.4 LoginCustomer 19
4.2.5 GetCustomersPlatform 19
4.2.6 DeleteCustomer 19
4.2.7 ChangePassword 20

5 Functions **21**
5.1 sum 21
5.2 mean 21
5.3 median 21
5.4 maximum 21
5.5 minimum 21

1 Introduction

There is a considerable amount of available Internet Of Things (IoT) and home automation technologies. Some are proprietary and some are open source, but its rare to offer full freedom and extensibility by design on very low-cost devices while providing ready-to-use products. Pancake framework should focus on this gap and offer very low cost and open devices based on ESP8266 chips. Services could prepared for home automation purpose and but the flexibility allows many other uses cases. These goals can be achievable thanks to modern languages and frameworks such as nodeJS and EcmaScript7. Public swagger API specification for Clashmore module could offer shallow learning curve for manufacturers creating compatible devices or offer iOS/Android UI for Clashmore thanks to Swagger auto generators. Pancake Service should offer SDK which will allow 3rd party developers writing universal and portable apps for any use cases easily. In the future, this could lead to extensions, an app marketplace and own ecosystem. Pancake Service SDK will be distributed as an npm package only, but in the future 3rd parties could offer iOS/Android SDKs as well. Good end-user adoption could be achieved by offering the Pancake Service as cloud service and as downloadable container for these who would prefer to host it locally. This architecture should allow pleasant experience and many possibilities for the end-user. The app developer can be independent from end-user and independent from the device manufacturer as well.

Security is a significant aspect of this project because embedded devices tend to be involved in many attacks. They often are low on resources and low on entropy. Some devices might have hardware random generators, but many embedded devices have resort to flawed pseudo random generators.

- Devices can be used against the owner because they are capable to actuate physical real-world objects.
- Devices can be used as a swarm in a distributed denial of service attack against remote target.

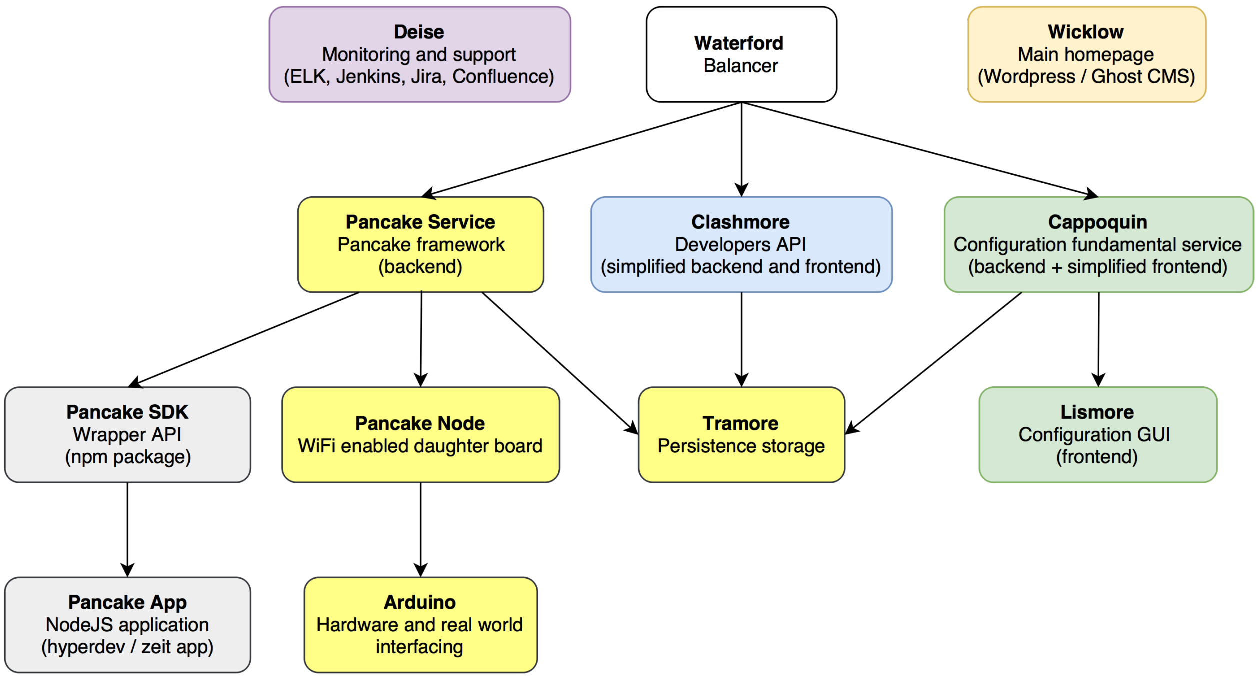


Figure 1: All modules involved in the Pancake project

Because of large scale and scope of this project as shown in the figure 1, for the formal specification I will only focus specifying a simplified subset of the project and describe only Pancake Service and Pancake Platform. PancakeService which consists of multiple instances of PancakePlaforms. Each platform is containing all the required Things and their connection between each other. Many of the Things are only sensors to sense the environment while some can be actuators altering the real world.

2 Global types and variables

$[CHAR]$
$[ROOM]$
$[LOCATION]$
$[TIME]$

$times : \text{iseq}\ TIME$

$BOOLEAN ::= true / false$

$toLower : CHAR \rightarrow CHAR$

$toLower = \{$
$`A` \mapsto `a`, `B` \mapsto `b`, `C` \mapsto `c`,$
$`a` \mapsto `a`, `b` \mapsto `b`, `c` \mapsto `c`,$
$`0` \mapsto `0`, `1` \mapsto `1`, `2` \mapsto `2`,$
$`@` \mapsto `@`, `.` \mapsto `.`, `\_` \mapsto `\_`...\}$

$charAlpha : \mathbb{P}\ CHAR$

$charAlpha = \{`a`...`z`, `A`...`Z`\}$

$charNum : \mathbb{P}\ CHAR$

$charNum = \{`0`...`9`\}$

3 Pancake Platform

3.1 PancakePlatform schema

PancakePlatform

$rooms : \mathbb{P}\, ROOM$
$roomLocations : LOCATION \pfun ROOM$
$things : THING \pfun LOCATION$
$allThings : \mathbb{P}\, THING$
$noiseThings, tempThings, lightThings, actuatorThings : \mathbb{P}\, THING$
$thingNames : THING \pfun \operatorname{seq} CHAR$
$minimums : THING \pfun \mathbb{Z}$
$thresholds : THING \pfun \mathbb{Z}$
$maximums : THING \pfun \mathbb{Z}$
$data : THING \pfun \operatorname{seq} \mathbb{Z}$
$edges : THING \rightarrow \operatorname{seq} THING$

$< noiseThings, tempThings, lightTHings, actuatorThings > \operatorname{partition} allThings$

$\operatorname{dom} edges \in allThings \setminus actuatorThings$
$\operatorname{ran} edges \in actuatorThings$

$\forall thing : \operatorname{dom} data \mid thing \in actuatorThings \bullet$
$\quad minimums(thing) \leq thresholds(thing) \leq maximums(thing)$

$\forall thing : \operatorname{dom} data \bullet$
$\quad (30 > \# thingNames(thing) > 3) \land$
$\quad (\forall index : dom(data(thing)) \bullet minimums(thing) \leq data(thing)(index) \leq maximums(thing))$

$\forall source : \operatorname{dom} edges$
$\quad \forall destination : edges(source) \bullet data(destination) = data(source) \land \# data(source) > 0$

$\operatorname{dom} thingNames = \operatorname{dom} minimums = \operatorname{dom} maximums = \operatorname{dom} things$

This is most important schema of the whole project, contains all the Things and keeps all their settings, connections between and history data. The thing could a physical sensor with internet connection implemented in hardware with ESP 2866 capable of sending data to the platform over WiFi. Depending what the Thing can do or what type it is they are split between consumers called actuators and between producers which sense real-life physical values as numbers and stored inside data set. If there is an entry in thingNames there is a guarantee that there is mapping in minimus and maximums as well.

- rooms contain all possible rooms in the system.
- roomLocations maps multiple specific locations inside the room.
- things are maped with their location inside the room.
- allThings contains all possible things in the system and is partitioned betwen noise, temperature, light and actuator Things.
- each Thing in the system has:
  - thingName a string with minimum lenght of 4 and maximum of 29.
  - minimum, maximum range constrains which limit the data inside these boundaries.
  - Their history data which logs all their value over time.
- Actuators on top of all the above properties has threshold information as well, but do not have mapping for edges. They can't connect any further Thing, because they are consuming the data and not passing it through.
- While non-actuators have list of edges containing information to what actuator things they are connected to. Edges can connect only to actuators things and they will share the same history data with connected Thing.

3.2 Operations

3.2.1 InitPancakePlatform

$$
\begin{array}{|l}
\hline
\textit{InitPancakePlatform} \\
\textit{PancakePlatform}' \\
\hline
\textit{roomLocations} =<> \\
\textit{things} =<> \\
\textit{thingNames} =<> \\
\textit{minimums} =<> \\
\textit{thresholds} =<> \\
\textit{maximums} =<> \\
\textit{data} =<> \\
\textit{edges} =<> \\
\hline
\end{array}
$$

InitPancakePlatform will initialise platform for the very first time with empty data

3.2.2 AddLocation

$$
\begin{array}{|l}
\hline
\textit{AddLocation} \\
\Delta \textit{PancakePlatform} \\
\textit{room}? : \textit{ROOM} \\
\textit{location}? : \textit{LOCATION} \\
\hline
\textit{location} \notin \operatorname{dom} \textit{roomLocations} \\
\textit{roomLocations}' = \textit{roomLocations} \cup \{\textit{location}? \mapsto \textit{room}?\} \\
\\
\textit{things}' = \textit{things} \\
\textit{thingNames}' = \textit{thingNames} \\
\textit{minimums}' = \textit{minimums} \\
\textit{thresholds}' = \textit{thresholds} \\
\textit{maximums}' = \textit{maximums} \\
\textit{data}' = \textit{data} \\
\textit{edges}' = \textit{edges} \\
\textit{rooms}' = \textit{rooms} \\
\textit{allThings}' = \textit{allThings} \\
\textit{noiseThings}' = \textit{noiseThings} \\
\textit{tempThings}' = \textit{tempThings} \\
\textit{lightThings}' = \textit{lightThings} \\
\textit{actuatorThings}' = \textit{actuatorThings} \\
\hline
\end{array}
$$

A room can contain multiple specific locations where the things could be placed, but location needs to be unique for each thing. All other information in the system will not change.

3.2.3 RemoveLocation

RemoveLocation

$$
\begin{array}{l}
\Delta PancakePlatform \\
location? : LOCATION \\
\hline
location \in \operatorname{dom} roomLocations \\
location \notin \operatorname{ran} things \\
roomLocations' = \{location?\} \lhd roomLocations \\
\\
things' = things \\
thingNames' = thingNames \\
minimums' = minimums \\
thresholds' = thresholds \\
maximums' = maximums \\
data' = data \\
edges' = edges \\
rooms' = rooms \\
allThings' = allThings \\
noiseThings' = noiseThings \\
tempThings' = tempThings \\
lightThings' = lightThings \\
actuatorThings' = actuatorThings
\end{array}
$$

RemoveLocation operation will remove only locations which are not yet associated with the thing. If there is already mapping made, you need to call RemoveThing from chapter 3.2.6 first. All other information in the system will not change.

3.2.4 AddThing

$$
\begin{array}{|l}
\textit{AddThing} \\
\hline
\Delta PancakePlatform \\
thing? : THING \\
location? : LOCATION \\
name? : seqCHAR \\
min? : \mathbb{Z} \\
max? : \mathbb{Z} \\
\hline
30 > \# \, name? > 3 \\
thing? \notin actuatorThings \\
thing? \notin \operatorname{dom} thingNames \\
location? \in \operatorname{dom} roomLocations \\
location? \notin \operatorname{ran} things \\
thingNames' = thingNames \cup \{thing? \mapsto name?\} \\
things' = things \cup \{thing? \mapsto location?\} minimums' = minimus \cup \{thing? \mapsto min?\} \\
maximus' = maximus \cup \{thing? \mapsto min?\} \\
edges' = edges \cup \{thing? \mapsto <>\} \\
\\
roomLocations' = roomLocations \\
thresholds' = thresholds \\
data' = data \\
rooms' = rooms \\
allThings' = allThings \\
noiseThings' = noiseThings \\
tempThings' = tempThings \\
lightThings' = lightThings \\
actuatorThings' = actuatorThings \\
\hline
\end{array}
$$

AddThing will append given thing, the call needs to be given the Thing itself, it's name, minimum and maximum ranges. Location needs to be unique, associated with a room, but not with any previous Thing. The thing can't be actuatorThing (for actuators there is call called AddActuatorThing). The name needs to be more than 3 characters and less than 30, The thing can't be in the system already. After all, requirements are meet, the platform will be updated with new thingName, minimums, maximums and empty sequence of edges associated with this Thing. All other information will stay same.

3.2.5 AddActuatorThing

AddActuatorThing

$\Delta PancakePlatform$
$thing? : THING$
$location? : LOCATION$
$name? : seqCHAR$
$min? : \mathbb{Z}$
$max? : \mathbb{Z}$
$threshold? : \mathbb{Z}$

$30 > \# name? > 3$
$thing? \in actuatorThings$
$thing? \notin \mathrm{dom}\ thingNames$
$location? \in \mathrm{dom}\ roomLocations$
$location? \notin \mathrm{ran}\ things$
$things' = things \cup \{thing? \mapsto location?\}$
$min? \leq threshold? \leq max?$
$thingNames' = thingNames^\frown < name? >$
$minimums' = minimus^\frown < min? >\}$
$thresholds' = thresholds^\frown < threshold? >$
$maximus' = maximus^\frown < min? >$
$edges' = edges \cup \{thing? \mapsto \varnothing\}$

$roomLocations' = roomLocations$
$data' = data$
$rooms' = rooms$
$allThings' = allThings$
$noiseThings' = noiseThings$
$tempThings' = tempThings$
$lightThings' = lightThings$
$actuatorThings' = actuatorThings$

AddActuatorThing will append a given thing, the call needs to be given the Thing itself, it's name, minimum and maximum ranges. Location needs to be unique, associated with a room, but not with any previous Thing. The thing needs to be actuatorThing (for non-actuators there is call called AddThing). The name needs to be more than 3 characters and less than 30, The thing can't be in the system already. After all, requirements are meet, the platform will be updated with new thingName, minimums, maximums and empty sequence of edges associated with this Thing. All other information will stay same.

3.2.6 AddData

AddData

$$
\begin{array}{l}
\Delta PancakePlatform \\
thing? : THING \\
time? : TIME \\
value? : \mathbb{Z} \\
\hline
thing? \in \operatorname{dom} thingNames \\
thing? \notin actuatorThings \\
time? \in times \\
minimums(thing?) \leq value? \leq maximums(thing?) \\
data' = data \oplus \{thing? \mapsto (time? \mapsto value?)\} \\
\\
roomLocations' = roomLocations \\
things' = things \\
thingNames' = thingNames \\
minimums' = minimums \\
thresholds' = thresholds \\
maximums' = maximums \\
edges' = edges \\
rooms' = rooms \\
allThings' = allThings \\
noiseThings' = noiseThings \\
tempThings' = tempThings \\
lightThings' = lightThings \\
actuatorThings' = actuatorThings
\end{array}
$$

AddData is called when a given Thing wants to store the value it measured. It is checked if the thing is in Platform and if the time is valid. The data value needs to be within the minimum and maximum ranges. All other information will stay the same. The data can be entered into the Platform only from non-Actuator Things. All other information will stay same.

3.2.7 RemoveThing

$$
\begin{array}{l}
\textit{RemoveThing} \\
\hline
\Delta \textit{PancakePlatform} \\
\textit{thing}? : \textit{THING} \\
\hline
\textit{thing}? \in \operatorname{dom} \textit{thingNames} \\
\textit{thingNames}' = \{\textit{thing}?\} \lhd \textit{thingNames} \\
\textit{things}' = \{\textit{thing}?\} \lhd \textit{thingNames} \\
\textit{minimus}' = \{\textit{thing}?\} \lhd \textit{minimus} \\
\textit{thresholds}' = \{\textit{thing}?\} \lhd \textit{thresholds} \\
\textit{maximus}' = \{\textit{thing}?\} \lhd \textit{maximus} \\
\textit{data}' = \{\textit{thing}?\} \lhd \textit{data} \\
\textit{edges}' = \{\textit{thing}?\} \lhd \textit{edges} \\
\\
\textit{roomLocations}' = \textit{roomLocations} \\
\textit{thingNames}' = \textit{thingNames} \\
\textit{minimums}' = \textit{minimums} \\
\textit{thresholds}' = \textit{thresholds} \\
\textit{maximums}' = \textit{maximums} \\
\textit{data}' = \textit{data} \\
\textit{edges}' = \textit{edges} \\
\textit{rooms}' = \textit{rooms} \\
\textit{allThings}' = \textit{allThings} \\
\textit{noiseThings}' = \textit{noiseThings} \\
\textit{tempThings}' = \textit{tempThings} \\
\textit{lightThings}' = \textit{lightThings} \\
\textit{actuatorThings}' = \textit{actuatorThings} \\
\hline
\end{array}
$$

RemoveThing requires only thing parameter and successfully removes it's mapping from thingNames, minimus, thresholds, maximums, data and edges. Will work with both non-actuator things and actuator things, but they need to be already in the system and have stored thingName, which will ensure that they have stored minimums, maximums as well. All other information will stay same.

3.2.8 ChangeThingRange

ChangeThingRange

$\Delta PancakePlatform$
$thing? : THING$
$min? : \mathbb{Z}$
$max? : \mathbb{Z}$

$thing? \in \operatorname{dom} thingNames$
$thing? \notin actuatorThings$
$data' = \{thing?\} \lhd data$
$data' = data \oplus \{thing? \mapsto <>\}$
$minimus' = minimus \oplus \{thing? \mapsto min?\}$
$maximus' = maximus \oplus \{thing? \mapsto max?\}$

$roomLocations' = roomLocations$
$things' = things$
$thingNames' = thingNames$
$thresholds' = thresholds$
$edges' = edges$
$rooms' = rooms$
$allThings' = allThings$
$noiseThings' = noiseThings$
$tempThings' = tempThings$
$lightThings' = lightThings$
$actuatorThings' = actuatorThings$

If the Thing was setup with wrong range and needs to be updated in the runtime then, ChangeThingRange can be called. It will require desired Thing and the new min and max values. The thing needs to be inside the system already and can't be used with actuatorThings (for these there is ChangeActuatorThingRange operation). The data for the thing needs to be cleared because the new range could invalidate the constrain of the existing data and probably this is the reason why the range needs to be updated.

3.2.9 ChangeActuatorThingRange

$$
\begin{array}{|l}
\textit{ChangeActuatorThingRange} \\
\Delta PancakePlatform \\
thing? : THING \\
min? : \mathbb{Z} \\
threshold? : \mathbb{Z} \\
max? : \mathbb{Z} \\
\hline
thing? \in \operatorname{dom} thingNames \\
thing? \in actuatorThings \\
data' = data \oplus \{thing? \mapsto <>\} \\
minimus' = minimus \oplus \{thing? \mapsto min?\} \\
thresholds' = thresholds \oplus \{thing? \mapsto threshold?\} \\
maximus' = maximus \oplus \{thing? \mapsto max?\} \\
\\
roomLocations' = roomLocations \\
things' = things \\
thingNames' = thingNames \\
edges' = edges \\
rooms' = rooms \\
allThings' = allThings \\
noiseThings' = noiseThings \\
tempThings' = tempThings \\
lightThings' = lightThings \\
actuatorThings' = actuatorThings
\end{array}
$$

If an actuator Thing was setup with wrong range and needs to be updated in the runtime, then ChangeThingRange can be called. It will require desired Thing and the new min and max values. The thing needs to be inside the system already and can't be used with non-actuatorThings (for these there is ChangeThingRange operation). The data for the thing needs to be cleared because the new range could invalidate the constrain of the existing data and probably this is the reason why the range needs to be updated.

3.2.10 FindWhen

$$
\begin{array}{|l}
\textit{FindWhen} \\
\Xi PancakePlatform \\
thing? : THING \\
val? : \mathbb{Z} \\
when! : \operatorname{iseq} TIME \\
\hline
thing? \in \operatorname{dom} data \\
when! = times \llparenthesis\, data(thing?)^{\sim} \llparenthesis val? \rrparenthesis \,\rrparenthesis
\end{array}
$$

To find all moments when the thing's data had specified value a FindWhen operation can be called. It doesn't change anything in the platform, only returns data. It will return a sequence of all times when the specific val was recorded. The thing needs to be in the system.

3.2.11 AddEdge

AddEdge

$$
\begin{array}{l}
\Delta PancakePlatform \\
source? : THING \\
destination? : THING \\
\hline
source? \in \mathrm{dom}\, thingNames \\
destination? \in \mathrm{dom}\, thingNames \\
source? \notin actuatorThings \\
destination? \in actuatorThings \\
\forall s : THING \mid s \in allThings \land s \notin actuatorThings \bullet destination? \notin edges(s?) \\
edges' = edges \oplus \{edges(source?)^\frown < destination? >\} \\
\\
roomLocations' = roomLocations \\
things' = things \\
thingNames' = thingNames \\
minimums' = minimums \\
thresholds' = thresholds \\
maximums' = maximums \\
data' = data \\
rooms' = rooms \\
allThings' = allThings \\
noiseThings' = noiseThings \\
tempThings' = tempThings \\
lightThings' = lightThings \\
actuatorThings' = actuatorThings
\end{array}
$$

Operation AddEdge is used to add to a non-actuator source thing an edge connection which is pointing to an actuator destination thing. Both source and destination things needs to be part of the system and the destination thing cannot be used as a destination edge by any other non-actuator things. Only one non-actuator thing can point to the actuator thing in the whole system. The operation will update the edges, but all other information will stay same.

3.2.12 RemoveEdge

RemoveEdge

$\Delta PancakePlatform$
$source? : THING$
$destination? : THING$

$source? \in \operatorname{dom} thingNames$
$destination? \in \operatorname{dom} thingNames$
$source? \notin actuatorThings$
$destination? \in actuatorThings$
$destination? \notin edges(source?)$
$edges' = edges \oplus squash(edges(source?) \rhd \{destination?\})$

$roomLocations' = roomLocations$
$things' = things$
$thingNames' = thingNames$
$minimums' = minimums$
$thresholds' = thresholds$
$maximums' = maximums$
$data' = data$
$rooms' = rooms$
$allThings' = allThings$
$noiseThings' = noiseThings$
$tempThings' = tempThings$
$lightThings' = lightThings$
$actuatorThings' = actuatorThings$

A RemoveEdge operation is useful to make sure a given source non-actuator thing is not using destination actuator thing as an edge. Both source and destination things needs to be part of the system. The operation will update the edges, but all other information will stay same.

3.2.13 GetThingStats

GetThingStats

$\Xi PancakePlatform$
$thing? : THING$
$timeStart? : TIME$
$timeEnd? : TIME$
$min! : \mathbb{Z}$
$avg! : \mathbb{Z}$
$med! : \mathbb{Z}$
$max! : \mathbb{Z}$

$\{timeStart?, timeEnd?\} \subseteq times$
$times^{\sim}(timeStart?) < times^{\sim}(timeEnd?)$
$thing? \in \operatorname{dom}(data)$
$\exists samples : \operatorname{seq} \mathbb{Z} \bullet samples = squash(times^{\sim}(timeStart?)...times^{\sim}(timeEnd?) \lhd data(thing?))$
$min! = minimum(samples) avg! = mean(samples)$
$med! = median(samples)$
$max! = maximum(samples)$

Statistical information can be gained with GetThingStats operation. It will not change anything in the platform, only return requested, min,max, avg and med results. The given parameter thing needs to be part of the system and the timeStart and timeEnd needs to be valid times and timeStart needs happen before the timeEnd. The operation is using the mathematical functions from chapter 5.4 to compute the statistical results on data which was stored between given time interval.

3.2.14 isThingOver

$$
\begin{array}{|l}
\textit{isThingOver} \\
\hline
\Xi PancakePlatform \\
thing? : THING \\
result! : BOOLEAN \\
\hline
time? \in times \\
thing? \in \mathrm{dom}(data) \\
thing? \in actuatorThings \\
\# \, data(thing?) > 0 \\
(data(thing?)(\# \, data(thing?)) > thresholds(thing?) \land result! = true) \lor \\
(data(thing?)(\# \, data(thing?)) \leq thresholds(thing?) \land result! = false) \\
\hline
\end{array}
$$

Use isThingOver operation to find out if given actuator thing is over the threshold. Operation requires the thing to have some data and it will return the state based on the very last data entry. And it will return BOOLEAN response if the thing at that time was over the threshold. Operation itself will not affect any information inside the platform.

3.2.15 isThingUnder

$$
\begin{array}{|l}
\textit{isThingUnder} \\
\hline
\Xi PancakePlatform \\
thing? : THING \\
result! : BOOLEAN \\
\hline
time? \in times \\
thing? \in \mathrm{dom}(data) \\
thing? \in actuatorThings \\
\# \, data(thing?) > 0 \\
(data(thing?)(\# \, data(thing?)) \leq thresholds(thing?) \land result! = true) \lor \\
(data(thing?)(\# \, data(thing?)) > thresholds(thing?) \land result! = false) \\
\hline
\end{array}
$$

Use isThingUnder operation to find out if given actuator thing is below, or at the threshold. Operation requires valid time when we the actuator was over the threshold, know the specific thing. And it will return BOOLEAN response if the thing at that time was below the threshold. Operation itself will not affect the state of the platform.

4 PancakeService

4.1 Schemas

4.1.1 Customer

Customer

$$
\begin{array}{|l}
name : \operatorname{seq} CHAR \\
email : \operatorname{seq} CHAR \\
password : \operatorname{seq} CHAR \\
\hline
70 \geq \# name \geq 2 \\
70 \geq \# email \geq 5 \\
\exists_1 char : CHAR \mid char \in \operatorname{ran} email \bullet char = \text{‘@‘} \\
\exists char : CHAR \mid char \in \operatorname{ran} email \bullet char = \text{‘.‘} \\
100 \geq \# password \geq 10 \\
\exists char : CHAR \mid char \in \operatorname{ran} password \bullet char \in charAlpha \\
\exists char : CHAR \mid char \in \operatorname{ran} password \bullet char \in charNum
\end{array}
$$

A Customer has stored his name and credentials, the name needs to be longer than 1 character and shorter than 71. Email needs to be at least 5 characters long and up to 70, 5 characters is the absolute minimum for email format ‘a@b.c‘, the email needs to contain one ‘@‘ character and at least one dot. The password needs to be longer than 9 and less than 101 while they needs to use at least one alphanumerical character and one numerical character.

4.1.2 PancakeService

PancakeService

$$
\begin{array}{|l}
customers : \operatorname{seq} Customer \\
platforms : \operatorname{seq} PancakePlatform \\
\hline
\# customers = \# platforms
\end{array}
$$

PancakeService holds all the customers and all their individual instances of their platforms. Both customers and platforms needs to be the same size.

4.2 Operations

4.2.1 InitPancakeService

$$
\begin{array}{|l}
\hline
\mathit{InitPancakeService} \\
\mathit{PancakeService}' \\
\hline
\mathit{customers} = <> \\
\mathit{platforms} = <> \\
\hline
\end{array}
$$

InitPancakeService operation will initialize empty sequences for customers and platforms.

4.2.2 RegisterCustomer

$$
\begin{array}{|l}
\hline
\mathit{RegisterCustomer} \\
\Delta \mathit{PancakeService} \\
\mathit{newCustomer}? : \mathit{Customer} \\
\mathit{emptyPlatform} : \mathit{PancakePlatform} \\
\hline
\mathit{newCustomer} \notin \mathit{customers} \\
\mathit{customers}' = \mathit{customers} \cup \mathit{newCustomer} \\
\mathit{platforms}' = \mathit{platforms} \cup \mathit{emptyPlatform} \\
\hline
\end{array}
$$

To add new customer a RegisterCustomer operation can be used. A newCustomer and emptyPlatform needs to be supplied. An existing customer can't register again. If desired customers can create a new customer account under a different email.

4.2.3 RegisteredCustomers

$$
\begin{array}{|l}
\hline
\mathit{RegisteredCustomers} \\
\Xi \mathit{PancakeService} \\
\mathit{count}! : \mathbb{N} \\
\hline
\mathit{count}! = \# \operatorname{dom} \mathit{customers} \\
\hline
\end{array}
$$

To find out how popular the Pancake is there is simple operation RegisteredCustomers. It will count how many existing customers the PancakeService has. The operation doesn't change any information in the service.

4.2.4 LoginCustomer

$$
\begin{array}{|l}
\hline \textit{LoginCustomer} \\
\Xi PancakeService \\
email? : \text{seq } CHAR \\
password? : \text{seq } CHAR \\
logged! : BOOLEAN \\
\hline
70 \geq \# email? \geq 5 \\
\exists_1 char : CHAR \mid char \in \text{ran } email? \bullet char = \text{‘@‘} \\
\exists char : CHAR \mid char \in \text{ran } email? \bullet char = \text{‘.‘} \\
100 \geq \# password? \geq 10 \\
\exists char : CHAR \mid char \in \text{ran } password? \bullet char \in charAlpha \\
\exists char : CHAR \mid char \in \text{ran } password? \bullet char \in charNum \\
\\
\exists e : \text{seq } CHAR \mid e = email? \mathbin{⨾} toLower \\
\\
(\exists_1 c : Customer \mid c \in customers \land c.email = e \land c.password = password? \bullet logged! = true) \\
\qquad \lor \\
(\nexists c : Customer \mid c \in customers \land c.email = e \land c.password = password? \bullet logged! = false) \\
\hline
\end{array}
$$

To authenticate existing customer a LoginCustomer operation can be called. Email needs to be at least 5 characters long and up to 70, 5 characters is an absolute minimum for email format ‘a@b.c‘, the email needs to contain one ‘@‘ character and at least one dot. The password needs to be longer than 9 and less than 101 while they needs to use at least one alphanumerical character and one numerical character. The email then will be converted to a lowCaps string. The operation will not change any state, only verify if there exists any customer who has same email and password, if yes it will return BOOLEAN true for logged. If there will be no single entry where email and password were matched it will return false.

4.2.5 GetCustomersPlatform

$$
\begin{array}{|l}
\hline \textit{GetCustomersPlatform} \\
\Xi PancakeService \\
loggedCustomer? : Customer \\
platform! : PancakePlatform \\
\hline
loggedCustomer \in customers \\
platform! = platforms(customers^{\sim}(customer)) \\
\hline
\end{array}
$$

Each customer has an own dedicated platform, use GetCustomersPlatform operation to find what platform is allocated to the loggedCustomer. Platform state is not modified, loggedCustomer is provided and returned platform.

4.2.6 DeleteCustomer

$$
\begin{array}{|l}
\hline \textit{DeleteCustomer} \\
\Delta PancakeService \\
customerToDelete? : Customer \\
\hline
customerToDelete \in customers \\
customers' = squash(customers \rhd \{customerToDelete\}) \\
platforms' = squash(customers^{\sim}(customerToDelete) \lhd platforms) \\
\\
\hline
\end{array}
$$

To unsubscribe from the service DeleteCustomer operation can be used with customerToDelete argument.

4.2.7 ChangePassword

$$
\begin{array}{|l}
\textit{ChangePassword} \\
\hline
\Delta \textit{PancakeService} \\
\textit{customer}? : \textit{Customer} \\
\textit{customerWithNewPassword}? : \textit{Customer} \\
\hline
\textit{customer} \in \textit{customers} \\
\textit{customers}' = \textit{customers} \oplus \{\textit{customers}^{\sim} (\!| \{\textit{customer}?\} |\!) \mapsto \textit{customerWithNewPassword}?\} \\
\textit{platforms}' = \textit{platforms} \\
\hline
\end{array}
$$

Operation ChangePassword requires customer and the same customer with updated new password. It will interchange the customer with the customerWithNewPassword inside the customers, while platforms will stay untouched.

5 Functions

5.1 sum

$$
\begin{array}{|l}
\mathit{sum} : \operatorname{seq} X \rightarrow \mathbb{Z} \\
\hline
\forall\, s : \operatorname{seq} X;\ x : \mathbb{Z} \bullet \\
\mathit{sum} <>= 0 \land \\
\mathit{sum} < H > {}^\frown T = H + \mathit{sum}(T)
\end{array}
$$

The sum will split the sequence into head and tail and recursively calculate sum of all members of the sequence.

5.2 mean

$$
\begin{array}{|l}
\mathit{mean} : \operatorname{seq}_1 \mathbb{Z} \rightarrow \mathbb{Z} \\
\hline
\forall\, \mathit{sequence} : \operatorname{seq}_1 \mathbb{Z} \bullet \mathit{mean}(\mathit{sequence}) = \mathit{sum}(\mathit{sequence}) \operatorname{div} \#\, \mathit{sequence}
\end{array}
$$

Mean will calculate the average from the sequence of the data by summing up the whole sequence and then diving by the size of the sequence.

5.3 median

$$
\begin{array}{|l}
\mathit{median} : \operatorname{seq}_1 \mathbb{Z} \rightarrow \mathbb{Z} \\
\hline
\forall\, \mathit{sequence} : \operatorname{seq}_1 \mathbb{Z} \bullet \\
\quad \exists_1\, \mathit{Historygram} : \mathbb{Z} \nrightarrow \mathbb{N}_1 \bullet \\
\qquad \operatorname{dom}(\mathit{Historygram}) = \operatorname{ran}(\mathit{sequence}) \land \\
\qquad (\forall\, \mathit{index} : \operatorname{dom}(\mathit{Historygram}) \bullet \mathit{Historygram}(\mathit{index}) = \#(\mathit{sequence} \rhd \{\mathit{index}\})) \land \\
\qquad \exists_1\, \mathit{median}(\mathit{sequence}) : \mathbb{Z} \mid \mathit{median}(\mathit{sequence}) \in \operatorname{dom}(\mathit{Historygram}) \bullet \\
\qquad\quad \mathit{Historygram}(\mathit{median}(\mathit{sequence})) = \mathit{max}(\operatorname{ran} \mathit{Historygram})
\end{array}
$$

Median is calculated with help of histogram. First the function will map how often each value was used, then it will find the value of the most used one.

5.4 maximum

$$
\begin{array}{|l}
\mathit{maximum} : \operatorname{seq}_1 \mathbb{Z} \rightarrow \mathbb{Z} \\
\hline
\forall\, \mathit{sequence} : \operatorname{seq}_1 \mathbb{Z} \bullet \mathit{maximum}(\mathit{sequence}) = \mathit{max}(\operatorname{ran} \mathit{sequence})
\end{array}
$$

The maximum is a simple function which calculates the maximum value from the range of the given sequence.

5.5 minimum

$$
\begin{array}{|l}
\mathit{minimum} : \operatorname{seq}_1 \mathbb{Z} \rightarrow \mathbb{Z} \\
\hline
\forall\, \mathit{sequence} : \operatorname{seq}_1 \mathbb{Z} \bullet \mathit{minimum}(\mathit{sequence}) = \mathit{min}(\operatorname{ran} \mathit{sequence})
\end{array}
$$

Minimum is a simple function which calculates the minimum value from the range of the given sequence.