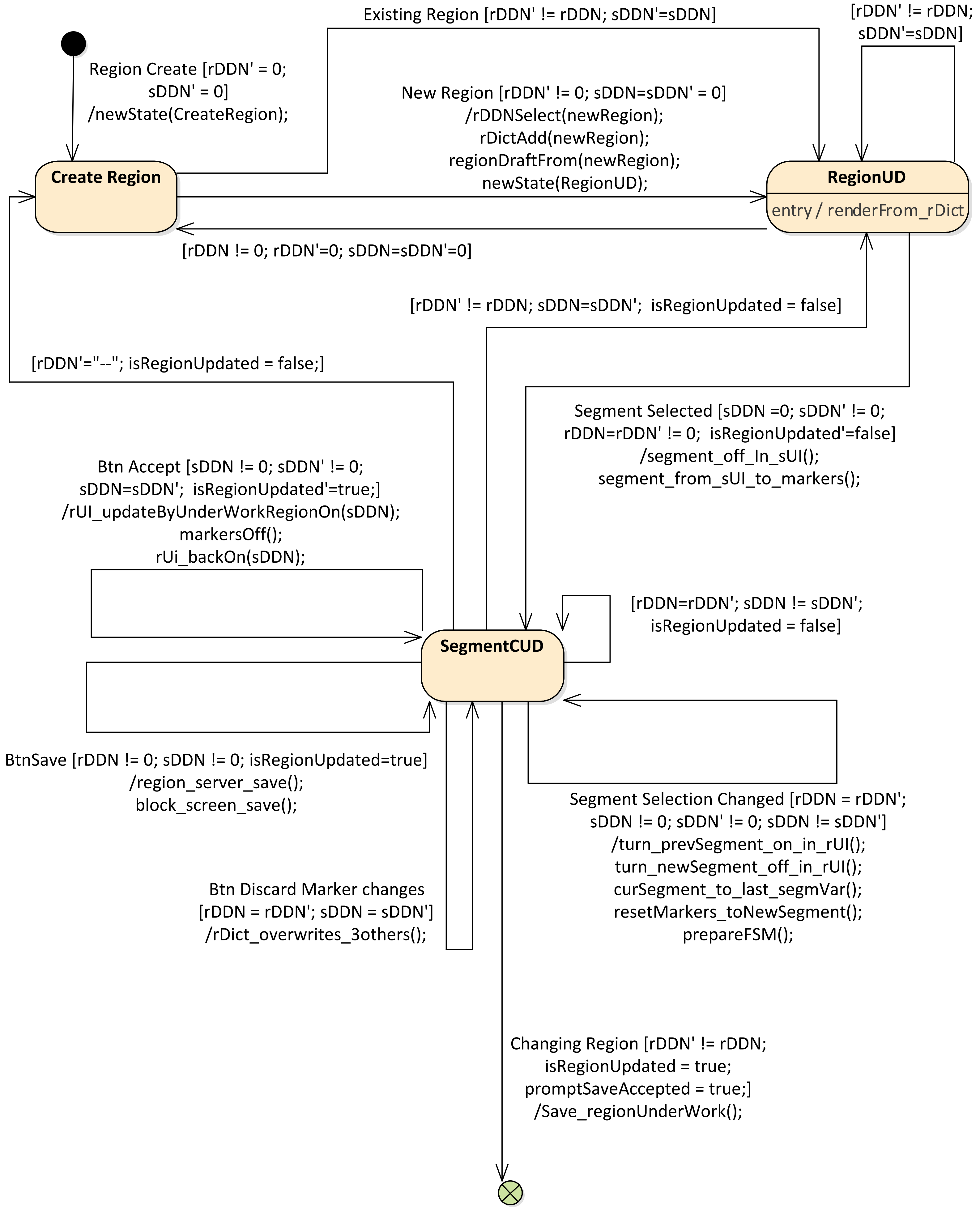
You are welcome to be against mathematical specification!

In fact many people indeed are even against the discussion on this topic. This may be related to a dissatisfaction with mathematics in general, or being too passionate for agile approach. I am convinced by experience, that starting by such an specification leads to the economy of cycles of agile life. It prevents logical errors in the fundation of the project. Starting from a robust foundation let's you take speed faster. Development will be much more smooth.

If you would like to still be against specification, you are welcome as before. If you got interested to investigate the usefulness of it further with me, then congratulations! We are starting a jurney in rigorous software development that let's the developer to breath stress-free during the development. Basic understanding of set theory is assumed. Having that basis, we can start right in the specification development.


Existing Region [rDDN' != rDDN; sDDN'=sDDN]
[rDDN' != rDDN;
sDDN'=sDDN]
Region Create [rDDN' = 0;
sDDN' = 0]
/newState(CreateRegion);
New Region [rDDN' != 0; sDDN=sDDN' = 0]
/rDDNSelect(newRegion);
rDictAdd(newRegion);
regionDraftFrom(newRegion);
newState(RegionUD);
Create Region
RegionUD
entry / renderFrom_rDict
[rDDN != 0; rDDN'=0; sDDN=sDDN'=0]
[rDDN' != rDDN; sDDN=sDDN'; isRegionUpdated = false]
[rDDN'="--"; isRegionUpdated = false;]
Segment Selected [sDDN =0; sDDN' != 0;
rDDN=rDDN' != 0; isRegionUpdated'=false]
/segment_off_In_sUI();
segment_from_sUI_to_markers();
Btn Accept [sDDN != 0; sDDN' != 0;
sDDN=sDDN'; isRegionUpdated'=true;]
/rUI_updateByUnderWorkRegionOn(sDDN);
markersOff();
rUi_backOn(sDDN);
[rDDN=rDDN'; sDDN != sDDN';
isRegionUpdated = false]
SegmentCUD
BtnSave [rDDN != 0; sDDN != 0; isRegionUpdated=true]
/region_server_save();
block_screen_save();
Segment Selection Changed [rDDN = rDDN';
sDDN != 0; sDDN' != 0; sDDN != sDDN']
/turn_prevSegment_on_in_rUI();
turn_newSegment_off_in_rUI();
curSegment_to_last_segmVar();
resetMarkers_toNewSegment();
prepareFSM();
Btn Discard Marker changes
[rDDN = rDDN'; sDDN = sDDN']
/rDict_overwrites_3others();
Changing Region [rDDN' != rDDN;
isRegionUpdated = true;
promptSaveAccepted = true;]
/Save_regionUnderWork();

```
── section SegmentCUD parents standard_toolkit
└──

!IMPORTANT:
before accepting any point in the "mouse input file", have the region rect reference,
check that the point clicked
always inside region


/* state: SegmentCUD */

/*
CUD:
Control, Update, Delete
*/

/* regionCDS is "only" toggled to specify the region corners update-state */
── theorem NoRegionUpdate
  ⊢? state = state′ = SegmentCUD ⇒ regionCDS = regionCDS′ = F
└──

──
  ON == T
└──
──
  OFF == F
└──

main DS:
- rDict
- sUI
- rUI
- RDraft: region draft, to which any region rect update or segment update is applied,
saving will send it to server
- segmentCDS: control variable, holds information that in SegmentCUD which segment
gets update
- regionCDS: control variable, holds information that in RegionUD, if region rect got
updated
- previousSegment: when sDDN selection changes, holds the previous value, by resaving
it every time
selection changes, right before using the old previousSegment, making
previousSegment' = sDDN'


functions:
  InitSegmentCDS(v?)
  OverwriteSUI(Region, segment_name, On/OFF)
  OverwriteSMarker(Region, segment_name, On/OFF)
  OverwriteSDraft(p1?, p2?, segment_name)
  let p1 = getSOrigin(rDict(region_name), segment_name)
  let p2 = getSTerminus(rDict(region_name), segment_name)


/* (1) */
/*

Init

Descr.:
1. Initialize SegmentCUD,
2. Region is already selected and not "--",
3. New segment selected and not "--",
4. previous Segment = --
5. Render the selected segment OFF, so to let user to manipulate markers, updating the
segment in draft, so:
6. turn on markers for the selected segment of selected region from original copy in rDict
7. reset the segment control dictionary all False

note:. the region-draft is already loaded with selected Region
(in RegionUD, only state which may navigate to SegmentCUD)

*/
┌─ SegmentCUD_Init
  ΔRegionEnv
  v?: VOID
|
  previousSegment = --
  sDDN = --

  sDDN′ ≠ --
  previousSegment′ = sDDN′

  rDDN = rDDN′
  rDDN′ ≠ --

  segmentCDS′ = InitSegmentCDS(v?)

  let region_name = rDDN′(rDDNVal′)
  let segment_name = sDDN′(sDDNVal′)

  ∃OverwriteSUI(NULL, segment_name, OFF)
  ∃OverwriteSMarker(rDict(region_name), segment_name, ON)
└─
```

```
/* (2) */
/*

Markers Updated

Descr.:

Overall:
When markers are updated, two things happen: a) draft's segment is overwrited, b)
segment CtrlDS updated

1. selected segment and region has no change
2. draft's segment is updated with the name of the current selected segment, using input
points of markers
3. segments' control DS gets updated based on the updated segment

Note:
- When mouse data available, two parties get it the same time: both the rFSM to setup
game objects, and
the OverwriteSDraft method to overwrite the draft. That's why the input of the method is
points, and not region.
- While user updates, markers are managed by rFSM
*/
┌─ SegmentCUD_MarkersUpdated
  ΔRegionEnv
  p1?, p2?: POINT
|
  rDDN = rDDN'
  sDDN = sDDN'

  let segment_name = sDDN(sDDNVal)

  ∃OverwriteSDraft(p1?, p2?, segment_name)
  segmentCDS' = segmentCDS ⊕ {segment_name ↦ T}
└─


/* (3) */
/*

Btn Discard

Descr.:

Purpose:
To undo marker changes in markers and turned-off sUI, using original Region copy of
rDict
```

Overall:
When Discard,

1. have segment's name to undo data for, and origin and terminus points of its original copy from rDict
2. using points and segment's name overwrite the draft
3. overwrite markers

*/

```
┌─ SegmentCUD_OnBtn_Discard
  ΔRegionEnv
|
  let segment_name = sDDN’(sDDNVal’)
  segmentCDS’ = segmentCDS ⊕ {segment_name ↦ F}

  let region_name = rDDN(rDDNVal)
  let p1 = getSOrigin(rDict(region_name), segment_name)
  let p2 = getSTerminus(rDict(region_name), segment_name)
  ∃OverwriteSDraft(p1, p2, segment_name)
  ∃OverwriteSMarker(rDict(region_name), segment_name, ON)
└─
```

/* (4) */
/*

Btn Commit

Descr.:

Purpose:
User being happy with what the segment must be, aims to submit changes for save

Overall:
Markers hold the state of manipulation, draft holds the same data, the sUI is overwritten by the draft, then
sUI turns on and markers turn off. Finally, sDDN'=-- so that we leave the segment.

Note:
When init back to SegmentCUD, if any segment is selected that its name points to True in the segmentCDS,
We turn on markers based on draft, not the rDaict

1.
2.
3.

```
*/
┌─ SegmentCUD_OnBtn_Commit
  ΔRegionEnv
|
  let segment_name = sDDN(sDDNVal)
  segmentCDS′ = segmentCDS ⊕ {segment_name ↦ T}

  ∃OverwriteSUI(RDraft, segment_name, ON)
  ∃OverwriteSMarker(NULL, NULL, OFF)

  sDDN′ = --
  state = SegmentCUD
  state′ = RegionUD
└─

/*
Leave save btn for RegionRD state
it is disabled in SegmentCUD
SegmentCUD state has only two btns:
Discard: undo changes on markers, overwrite sUI and draft back from rDict
Commit: accepts changes of draft (will not overwrite it), overwrites sUI of related
segment using draft
*/

/* (5) */
/*

Btn Save

Descr.:
If there exist any segment updated, screen is blocked and data sent to server, else, no
reaction

Note:
RDraft is sent to server to save, result will be back as a newly saved region, which will
replace rDict and
get selected rDDN and make sDDN'=-- and reset sUI, rDict and its newly saved copy also
updates draft,
which (this draft-updating) happens in the event handler that responds to the savedEvt
from server.

*/
┌─ PFSegmentCUD_OnBtn_Save
  ΞRegionEnv
|
  ∀ s : segmentCDS′ • second s = F
  state′ = state = SegmentCUD
```

```
└─

┌─ PTSegmentCUD_OnBtn_Save
  ΞRegionEnv
|
  ∃ s : segmentCDS′ • second s = T
  state′ = state = SegmentCUD
└─

──
  SegmentCUD_OnBtn_Save ==
          (PTSegmentCUD_OnBtn_Save ∧ SaveServer_BlockScreen) ∨
PFSegmentCUD_OnBtn_Save
└─

/* rDDN change in SegmentCUD state */

/*
we block scene to save draft-region and on resp of server the updated region is selected
and reRendered
so  because user Commited prompt-save, we are considered of nothing but make save
request and block
*/
── theorem promptSaveAccepted_axiom
  ⊢? promptSaveAccepted?: BOOL
└─

┌─ PTSegmentCUD_DrDDN
  ΔRegionEnv

|
  rDDN ≠ rDDN′
  sDDN = sDDN′ ≠ --

  ∃ s : segmentCDS′ • second s = T

  state = SegmentCUD
  (state′ = RegionUD ∧ rDDN′ ≠ --) ∨  (state′ = CreateRegion ∧ rDDN′ = --)
└─

┌─ PFSegmentCUD_DrDDN
  ΔRegionEnv

|
  rDDN ≠ rDDN′
  sDDN = sDDN′ ≠ --
  ∃ s : segmentCDS′ • second s = F

  state = SegmentCUD
  (state′ = RegionUD ∧ rDDN′ ≠ --) ∨  (state′ = CreateRegion ∧ rDDN′ = --)
└─

──
  SegmentCUD_DrDDN ==
      (PTSegmentCUD_DrDDN ∧ promptSaveAccepted? ∧ SaveServer_BlockScreen) ∨
PFSegmentCUD_DrDDN
└─

┌─ SaveServer_BlockScreen
  ΔRegionEnv
|
  ∃ReqServerSaveDraftEvt
  ∃ScreenBlockedWaiting
└─
```

/* sDDN change in SegmentCUD state */
/* if segment is updated, it means user commited changes from draft (markers' change are there) to sUI
so when changing sDDN, we have either updated sUI or original one (equal to rDict related region)
therefore we are not concerned on this, and simply turn sUI of leaving state on, save new state as prev.
and turn markers off
*/
D: Delta
If user did not commit changes and tried changing segment, we consider existing changes to be discarded
If the sDDN′ = -- then we turn off SMarker, because in that state marker manipulation is not allowed
while rDDN has no change, sDDN changed to either -- or something other than that
if the previous segment was not updated, we overwrite the sUI for it, using the unmanipulated rDict copy and
turning it on, else, with the knowledge that the previous segment was updated, we overwrite sUI with latest
update, which is kept in RDraft, and then turn sUI on

if the new segment that is obtained from sDDN' value (the current value of sDDN) has not been updated then
turn the markers on for it, using rDict original copy of the region and segment, where region name comes from
rDDN=rDDN', if the name of newly selected segment not found in segmentCDS, it could be only in a single case

when the value is "--", and in that case marker manipulation is not allowed, so turn markers off (and disallow
saving, by evaluating "segment's name ≠ "--").
Otherwise (which means that there is a segment_name with value of updated as true), turn on markers for the
newly selected segment, using RDraft.

```
┌─ SegmentCUD_DsDDN
  ΔRegionEnv
|
  rDDN = rDDN'
  sDDN ≠ sDDN' ≠ -- ∨ sDDN ≠ sDDN' = --

  let region_name = rDDN(rDDNVal)
  if ∃ s: segmentCDS • first s = previousSegment ∧ second s = F then
∃OverwriteSUI(rDict(region_name), previousSegment, ON)
  else ∃OverwriteSUI(RDraft, previousSegment, ON)

  previousSegment' = sDDN'(sDDNVal')
  let segment_name = sDDN'(sDDNVal')

  if ∃ s: segmentCDS • first s = segment_name ∧ second s = F then
∃OverwriteSMarker(rDict(region_name), segment_name, ON)
  else if ∀ s: segmentCDS • first s ≠ segment_name ∃OverwriteSMarker(NULL,
segment_name, OFF)
  else ∃OverwriteSMarker(RDraft, segment_name, ON)
└─
```

```
─ section AppData FSM - Home-DDN-Img parents standard_toolkit
└─

/* Announce for everyone: AppData is renamed to AppDS, being the main Data Structure
*/

/* seqURL, url and texture: related to interaction of HomeDDN and HomeImg*/

/* Given sets */
─ [ URL, TEXTURE, EVENTS ] └─

/* REQ target is the functionality that AppServiceProvider (abbr. AppProvider) offers */
─
  REQ ::= ReqLastUrls | ReqTexture
└─

/* RES is the response of the server that comes back to App, via server >JS >proxy
>provider */
─
  RES ::= LastUrlsCB | TextureCB
└─

/* Events to App */
─ theorem evtIn
  ⊢? {HInitEvt, SetTextureEvt} ⊆ EVENTS
└─

/* Events from App */
─ theorem evtOut
  ⊢? {UrlsUpdateEvt} ⊆ EVENTS
└─

/* State-schema and Init operation */
┌─ AppData
  urlsList: seq URL
  lastReqUrl: URL
  texture: TEXTURE
|
  lastReqUrl ≠ ∅ ⇔ (urlsList ≠ ∅ ∧ lastReqUrl ∈ ran urlsList)
└─

┌─ AppDataInit
  AppData
  req!: REQ
|
  urlsList = ∅
  lastReqUrl = ∅
  texture = Ø
  req! = ReqLastUrls
└─

/* Operations */

┌─ AppOnHomeInit
  ΞAppData
  e!,e?: EVENTS
|
  e? = HInitEvt
  e! = UrlsUpdateEvt
  urlsList! = urlsList = Ø
  lastReqUrl! = lastReqUrl = Ø
  texture! = texture = Ø
└─

┌─ AppOnLastUrlsCB
  ΔAppData
  urls?: seq URL
  rs?: RES
  e!: EVENTS
  rq!: REQ
  rq_url!: URL
|
  rs? = LastUrlsCB

  urlsList = Ø
  lastReqUrl = Ø
  texture = Ø

  urls? ≠ Ø
  urlsList' = urls?
  lastReqUrl' = head urls?
  texture' = texture

  rq! = ReqTexture
  rq_url! = lastReqUrl'
  e! = UrlsUpdateEvt
└─

┌─ AppOnTextureCB
  ΔAppData
  rs?: RES
  texture?: TEXTURE
  url?: URL
  e!: EVENTS
```

```
|
  rs? = TextureCB
  texture? ≠ Ø
  url? = lastReqUrl
  texture′ = texture?
  e! = UrlsUpdateEvt
└─
```

/* Image upload will trigger the same functionality, after uploaded image url returns to App */

```
┌─ AppOnSetTextureEvt
  ΔAppData
  e?: EVENTS
  url?: URL
  rq!: REQ
  url!: URL
|
  e? = SetTextureEvt
  url? ≠ lastReqUrl
  lastReqUrl′ = url?
  rq! = ReqTexture
  url! = url?
└─
```

```
— section appregion parents standard_toolkit
└─

/* Region scene */

/* Holds the Regions related data structures */
— [ URL, TEXTURE, RNAME ] └─

/* Boolean definition and meaning */
— theorem d_HasIntegerType
  ⊢? d : ℤ
└─
—
  F == (d ∈ ℤ ∧ d ∉ ℤ)
└─
—
  T ==  (d ∈ ℤ ∨  d ∉ ℤ)
└─
—
  BOOL ::= T | F
└─



—
  RSTATE ::= INITR | RC | RR | RU | RD
└─
/* So that if rState is in RR, mState can leave initM and be in any of other states, until
that sits in initM */
—
  MSTATE ::= INITM | RMC | RMR | RMU | RMD
└─
—
  REQ ::= ReqRegionsOfUrl
└─
—
  RES ::= RegionsOfUrlCB
└─
—
  INTENT ::= RadioRCreateN | DdnRReadN | RadioRUpdateN | BtnRDeleteN
└─
—
  UICMD ::= RadioRCreate | RadioRRead | RadioRUpdate | BtnRDelete
└─



—
  REPORT ::= WaitingServerResponse
```

```
└─

┌─ RegEnv
  rState: RSTATE
  mState: MSTATE
  rDDN: seq RNAME
  rDict:  RNAME ⤔ Region
  appImg: TEXTURE
  appUrl: URL
  uMagnitude: ℕ
  SceneLocked: BOOL
  interactable: BTN ⟶ BOOL
  promptSave: BOOL
|
  ran rDDN = dom rDict
  uMagnitude > 0
  appImg ≠ Ø
  appUrl ≠ Ø
└─

┌─ RegEnvInit
  RegEnv'
  rq!: REQ
  appImg?: TEXTURE
  appUrl?: URL
  uMagnitude?: ℕ
  rp!: REPORT
|
  rq! = ReqRegionsOfUrl
  rp! = WaitingServerResponse
  SceneLocked' = T
  promptSave' = F

  rState' = INITR
  mState' = INITM

  rDDN' = Ø
  rDict' = Ø

  appImg' = appImg?
  appUrl' = appUrl?
  uMagnitude' = uMagnitude?
  interactable = {(RadioRCreate, T), (DdnRRead, T | RadioRUpdate | BtnRDelete}
└─

/* when user in Home at chose img, RegionInitEvt to AppRegion, to JS query url-regions */
/* so when user goes to RegionScene, regions related to chosen url must already be in
```

```
place */

/* response of server: empty rDDN and rDICT, or, non-empty, each an op-schema */

/* AppRegion only sends regions set of imgUrl, when server response comes back. No init
data to be
ever sent! */

/*
# 1
lock region landing page until server response is in
after that, wait for opState change between create or read (select) region
read (select) region can lead to => update-region, delete-region or m-crud
*/

 ┌─ RegEnvUnLocked
  ΔRegEnv
  rs?: RES
  rDDN?: seq RNAME
  rDict?:  RNAME ⤔ Region
|
  rs? = RegionsOfUrlCB
  rDDN' = rDDN?
  rDict' = rDict?
  SceneLocked' = F
└─

┌─ NAME
  DECLS
|
  PREDS
└─
```

```
— section appregion2 parents standard_toolkit
└─

/* Region scene */

/* Holds the Regions related data structures */
— [ URL, TEXTURE, RNAME, GAMEOBJECT, VOID ] └─

/* Boolean definition and meaning */
— theorem d_HasIntegerType
  ⊢? d : ℤ
└─
—
  F == (d ∈ ℤ ∧ d ∉ ℤ)
└─
—
  T ==  (d ∈ ℤ ∨  d ∉ ℤ)
└─
—
  BOOL ::= T | F
└─

/* Region-Delete happens by BtnRegionDelete, Region-Update: l-r-click */

/* there will be a tmpGameObjArr for temporarily saving obj, will be reset on every
app-state-change */

/* Region-Update: when starts, resets tmpGameObjArr redefine to how many needed,
reset mFSM to init */

—
  STATE ::= CreateRegion | RegionUD | SegmentCUD
└─
/* this is memory-state, to organize mFSM, or memory-FSM */
—
  MSTATE ::= MInit | Origin | Terminus
└─
—
  MEASUREMENT ::= A1 | A2 | B1 | B2 | C1 | C2 | D1 | D2 | --
└─
—
  MSET ::= A1 | A2 | B1 | B2 | C1 | C2 | D1 | D2
└─
—
  PROMPT ::= AwaitServerUpdate | InvalidUpdate_LetUserRegRepeat |
PleaseProvideRegionName
                        | PleaseProvide_UniqueRegionName
```

```
└─
──
  validRegionName == RNAME ⟶ BOOL
└─
──
  evaluateState == (ℕ × MEASUREMENT) ⇸ VOID
└─
──
  resetTmpGObjArr == seq GAMEOBJECT ⤖ seq GAMEOBJECT
└─


┌─ Angles
  E1, E2: ℕ
|
  E1≥0
  E2≥0
└─

/* the image is loaded in the 3rd section of XY coordinate system */
┌─ POINT
  x,y: ℤ
|
  x < 0
  y < 0
└─

┌─ SEGMENT
  p1,p2: POINT
|
  p1 ≠ p2
└─

┌─ Region
  diagonal: SEGMENT
  angles: Angles
  mSet: MSET ⤔ SEGMENT
└─

/* Region Environment State Schema */
┌─ RegLabEnv
  mState: MSTATE
  rDDN: seq RNAME
  sDdnVal: MEASUREMENT
  cDdnVal: MEASUREMENT
  rDdnSelectedVal: ℤ
  rDict:  RNAME ⤖ Region
  segmentsObjDict: RNAME ⤖ seq (seq GAMEOBJECT)
  anglesObjDict: RNAME ⤖ seq GAMEOBJECT
  regionsObjDict: RNAME ⤖ seq GAMEOBJECT

  tmpGameObjArr: seq GAMEOBJECT

  appImg: TEXTURE
  appUrl: URL
  uMagnitude: ℕ
  SceneLocked: BOOL
|
  rDDN = dom rDict
  dom segmentsObjDict = dom rDict
  dom anglesObjDict = dom rDict
  dom regionsObjDict = dom rDict
  uMagnitude > 0
  appImg ≠ ∅
  appUrl ≠ ∅
  rDdnSelectedVal ≥ 1
└─

── section Initialization - Lock screen awaiting server update of Regions for current URL └─

/* Region Environment Initialization State Schema */
/* Lock the view totally, await update of Regions data structure of the current url: appUrl?
┌─ Init
  RegLabEnv′
  appImg?: TEXTURE
  appUrl?: URL
  uMagnitude?: ℕ
  p!: PROMPT
|
  mState′ = MInit
  rDDN′ = {(1, --)}
  rDdnSelectedVal′ = 1
  sDdnVal′ = cDdnVal′ = --
  rDict′ = ∅
  segmentsDict′ = ∅
  anglesDict′ = ∅
  appImg′ = appImg?
  appUrl′ = appUrl?
  uMagnitude′ = uMagnitude?
  tmpGameObjArr′ = ∅
  SceneLocked′ = T
  p! = AwaitServerUpdate
└─
```

```
/* WaitingServerUpdate maybe already obtained and cached in the AppRegion */


— section RegionEnv state conditions └─
/* code: any change to rDDN or sDDN must call a method that will reset state,
this is where to start programming!
*/

┌─ StateIsCreateRegion
  ΞRegLabEnv
|
  rDdnSelectedVal = 1
  sDdnVal = --
└─

┌─ StateIsRegionUD
  ΞRegLabEnv
|
  rDdnSelectedVal > 1
  sDdnVal = --
└─

┌─ StateIsSegmentCUD
  ΞRegLabEnv
|
  rDdnSelectedVal > 1
  sDdnVal ≠ --
└─

┌─ StateUpdate
  ΔRegLabEnv
|
  rDdnSelectedVal ≠ rDdnSelectedVal′ ∨ sDdnVal ≠ sDdnVal′
  reEvaluateState(rDdnSelectedVal′, sDdnVal′)
└─

/* There is the update of Regions data structure - a dictionary. Using it, deduce other DSs
*/
┌─ OnOkServerUpdate
  ΔRegLabEnv
  appUrl?: URL
  rDict?:  RNAME ⤖ Region
|
  appUrl = appUrl?
  rDDN′ = {(1, --)} ∪ dom rDict?
  rDict′ = rDict?
  SceneLocked′ = F
```

```
└─

/* Rendering function could be called any time an update to rDict is made (from server or
by the user) */
/* Using conjuction, include this schema in a robust def. of any schema when reRender is
needed */
┌─ Render
  ΔRegLabEnv
|
  anglesObjDict′ = anglesObjDict ⊕ {r: rDict′ • renderRegionAngles(r)}
  segmentsObjDict′ = segmentsObjDict ⊕ {r: rDict′ • renderRegionSegments(r)}
  regionsObjDict′ = regionsObjDict ⊕ {r: rDict′ • renderRegionMarkers(r)}
└─

┌─ OnBadOrNoServerUpdate
  ΞRegLabEnv
  p!: PROMPT
|
  p! = InvalidUpdate_LetUserRegRepeat
└─


/* Unsuccessful initialization ends up with a locked screen until a successful update from
server */
/* The successful update brings the Regions data structure from server to client */
──
  ServerUpdate == (OnOkServerUpdate ∧ Render) ∨ OnBadOrNoServerUpdate
└─

── section -R- CRUD operations after successful initialization └─


┌─ EnterCreateRegionState
  ΔRegLabEnv
  ΞStateIsCreateRegion
|
  [(rDdnSelectedVal ≠ 1 ∧ rDdnSelectedVal′ = 1 ∧ sDdnVal′ = sDdnVal = --)
        ∨ (rDdnSelectedVal′ = rDdnSelectedVal = 1 ∧ sDdnVal ≠ -- ∧ sDdnVal′ = --)]
  tmpGameObjArr′ = ∅
  mState′ = MInit
└─

/* tmpGameObjArr:
{txtA, txtB, txtC, LineImg} A and B for origin-terminus, C for region marker, LineImg:
segment
*/
┌─ CreateRegionDiagonalOrigin
```

```
  ΔRegLabEnv
  ΞStateIsCreateRegion
  p1?: POINT
|
  mState = MInit
  mState' = Origin
  validRegionVertex(p1?)
  tmpGameObjArr' = tmpGameObjArr ⊕ 1↦textObjFrom(p1)
└─

┌─ CreateRegionDiagonalTerminus
  ΔRegLabEnv
  ΞStateIsCreateRegion
  p2?: POINT
  p!: PROMPT
|
  mState = Origin
  mState' = Terminus
  validRegionVertex(p2?)
  let p1 = getVector2D(tmpGameObjArr(1))
  tmpGameObjArr' = tmpGameObjArr ⊕ {2↦textObjFrom(p2),
3↦regionMarkerFrom(p1,p2)}
  SceneLocked' = T
  p! = PleaseProvideRegionName
└─

┌─ CreatedRegionSave
  ΞRegLabEnv
  ΞStateIsCreateRegion
  p!: PROMPT
  rName?: RNAME
|
  SceneLocked = T
  p! = AwaitServerUpdate
└─

┌─ CreatedRegionSave_FailedOnDuplicatedName
  ΞRegLabEnv
  ΞStateIsCreateRegion
  p!: PROMPT
|
  SceneLocked = T
  p! = PleaseProvide_UniqueRegionName
└─

──
  CreatedRegionSave_RepeatOnDuplicatedName == CreatedRegionSave
```

```
└─

/* selectNewCreatedRegion(rDDN) will happen before rendering, it causes rDDN-refresh,
which calls Render */
┌─ CreatedRegionSuccess
  ΔRegLabEnv
  ΞStateIsCreateRegion
|
  tmpGameObjArr' = resetTmpGObjArr(tmpGameObjArr)
  updateWithNewRegionDS(rDict)
  selectNewCreatedRegion(rDDN)
└─

/* reRender region game obj dict (as rDict is updated, running the already created render
schema does the job) */
/* Render will be realized from selectNewCreatedRegion(rDDN) of CreatedRegionSuccess,
as it will cause rDDN refresh */
──
  R_CreatedRegion == (CreatedRegionSuccess ∧ Render) ∨
CreatedRegionSave_FailedOnDuplicatedName
└─


/* State: SegmentCUD */
/* crud-measures and reRendering */
```

```
— section appregion2text parents standard_toolkit
└

/*
Existence effect:
  rState: Environment state, being either in creation mode or read-update-delete, read ->
measure
  mState: either in '--', which is 'read mode of rState', or not '--', where M-CRUD happens
for a region
  rDDN: the list of region names in DDN
  rDict:  a function from region name to region data structure
  deleted this one:  toUpdate: a function from region name to boolean value,
                        to define if the region DS is overwritten and must get updated
  appImg: the image that the current region set belongs to
  appUrl: the url of the image described above
  uMagnitude: the value of the unit vector to apply (todo: apply a new unit value to all
regions' segments)
  rDdnSelectedVal: value of the current DDN selection, always equal to 1, at init (selected
= '--')
  SceneLocked: if scene is covered with a dialogue panel to communicate with user
*/



/* By here, we have regions DS either empty or not, meaning rDDN has either 1 dummy
element "--" or more
   and rDdnSelectedVal = 1, AND rendering phase is over with segments, region-markers
and angles, we may:
   - create region and select it
   - if #rDDN>1 then read any of its entries, from there ready to update/delete, having
Measure ≠ "--"
   - if rDDN is selected to other than dummy element, then can change "Measure" from
"--" to "A1, etc." and do
        measurement
   - if rDDN is selected to other than dummy element, and "Measure" is on "--", then can
update region,
        or delete it

        main DSs: segmentsObjDict, anglesObjDict, regionsObjDict
*/
```

```
— section AppData FSM - Unit parents standard_toolkit
└

— [ POS, URL, DATE, EVENTS ] └

/* Boolean definition and meaning */
— theorem d_HasIntegerType
  ⊢? d : ℤ
└
—
  F == (d ∈ ℤ ∧ d ∉ ℤ)
└
—
  T ==  (d ∈ ℤ ∨  d ∉ ℤ)
└
—
  BOOL ::= T | F
└

/* REQ target is the functionality that AppServiceProvider (abbr. AppProvider) offers */
—
  REQ ::= ReqLastUint | ReqSaveUnit
└

/* RES is the response of the server that comes back to App, via server >JS >proxy
>provider */
—
  RES ::= LastUintSavedCB | LastUnitResCB
└

/* Events to App */
— theorem evtIn
  ⊢? {UInitEvt, SetUnitEvt} ⊆ EVENTS
└

/* Events from App */
— theorem evtOut
  ⊢? {UnitUpdateEvt, UnitAvailableEvt} ⊆ EVENTS
└

/* State-schema and Init operation */

┌─ AppUnit
  magnitude: ℕ
  p1, p2: POS
  url: URL
  date: DATE
  isDirty: BOOL
|
  magnitude = 0 ⇔ (p1 = Ø ∧ p2 = Ø ∧ url = Ø ∧ date = Ø)
  magnitude ≠ 0 ⇔ (p1 ≠ Ø ∧ p2 ≠ Ø ∧ url ≠ Ø ∧ date ≠ Ø)
└─

┌─ AppUnitInit
  AppUnit′
  rq!: REQ
|
  magnitude′ = 0
  p1′ = Ø
  p2′ = Ø
  url′ = Ø
  date′ = Ø
  isDirty′ = F

  rq! = ReqLastUint
└─

/* Operations */

/* when user leaves unit scene, if unit created magn>0, appUnit sends srv evt sets
dirty=true
if a CB comes bk srv eql to unit, dirty sets false and user may nav to regions, else region
disabled
*/

┌─ AppOnSetUnitEvt
  ΔAppUnit
  e!, e?:EVENTS
  magnitude!, magnitude?: ℕ
  p1!, p2!, p1?, p2?: POS
  url!, url?: URL
  date!, date?: DATE
|
  e? = SetUnitEvt
  e! = UnitUpdateEvt

  magnitude? > 0
  magnitude′ = magnitude?
  p1! = p1′ = p1?
  p2! = p2′ = p2?
  url! = url′ = url?
  date! = date′ = date?
  isDirty′ = T

└─
```

```
— section new1spec parents standard_toolkit
└─

This specification describes state of App (scenes and data), home, unit, and region
scenes.
Use case: measure a unit, measure images, export CSV file of measurements.
App, means Prj, meaning the Singleton Project Prefab of Zenject

/* Type definitions */
— [ URL, TEXTURE, VPOS, EVENT, UI, CHAR, SERVEROPS ] └─

 /* Application Events */

 /* implementation: package authS and authF to authUpdateEvt
 /* events distributed for s:set meaning the event is betwen 's' and 'app' either as req. or
res.
 or related to a concept like Authentication, that could be between any 's' and 'app'.
 */
—
  AuthEvents ::= authSuccessEvt  | authFailEvt | qAuthStateEvt | reqLoginEvt
└─
—
  HomeEvents ::= hInitEvt | hDdnUpdateEvt
└─
—
  UnitEvents ::= uInitEvt | qUnitEvt | unitUpdateEvt
└─
—
  ImgUrlEvents ::=  reqTextureByUrlEvt
└─
—
  EVENTS == EVENT ∪ AuthEvents ∪ UnitEvents ∪ HomeEvents
└─


/* Boolean definition and meaning */
— theorem xBool_HasIntegerType
  ⊢? xBool : ℤ
└─
—
  F == (xBool ∈ ℤ ∧ xBool ∉ ℤ)
└─
—
  T ==  (xBool ∈ ℤ ∨  xBool ∉ ℤ)
└─
—
  BOOL ::= T | F
```

SHOW == T

HIDE == F

VISIBILITY ::= SHOW | HIDE

UITYPE ::= inField | btn | ddn

PANEL == UI ⇸ UITYPE

MNAME == A1 | A2 | B1 | B2 | C1 | C2 | D1 | D2 | UN

SCENE == HOME | UNIT | REGION

ENABLE == T

NAME == F

INTERACTIBILITY ::= enable | disable

ISINTRACTABLE == UITYPE ⇸ INTERACTIBILITY

TEXT == seq CHAR

UITEXT == UITYPE ⇸ TEXT

```
  ASYNQUEUE ::= getLatestUrls | getTextureOfUrl
└─




/* Schema types */
┌─ MVector
  p1, p2: VPOS
  uvMagn: ℕ
  name: MNAME
└─

┌─ Img
  url: URL
  texture: TEXTURE
└─

/* App State Schemas */

┌─ Signal
  FiredEvt: EVENTS ⟶ T
└─

┌─ AppImgUrl
  curImg: Img
  urls: seq URL
└─

┌─ AppImgUrlInit
  AppImgUrl′
|
  curImg′ = ∅
  urls′ = ∅
└─

┌─ AppUnit
  unitLatest′: MVector
  unitImg′: Img
└─

┌─ AppUnitInit
  AppUnit′
|
  unitLatest′ = ∅
  unitImg′ = ∅
```

```
└─

┌─ AppData
  loggedIn: BOOL
  scene: SCENE
  hDDN: seq URL
└─
┌─ AppDataInit
  AppData'
|
  loggedIn' = FALSE
  scene' = HOME
  hDDN = Ø
└─

┌─ AppProvider
  doLogin: EVENTS ⇸ BOOL
  asyncIntentQueue: ASYNQUEUE ⇸ SERVEROPS
  lastReqUrl: URL
└─
┌─ AppProviderInit
  AppProvider'
|
  doLogin' = Ø
  dom asyncIntentQueue' = Ø
  lastReqUrl' = Ø
└─

┌─ Authenticated
  AppData
|
  loggedIn = T
└─

┌─ UnAuthenticated
  AppData
|
  loggedIn = F
└─


┌─ AppInit
  AppImgUrlInit
  AppUnitInit
  AppDataInit
  HomeInit
  AppProviderInit
```

```
└─

/* Home State Schemas */

┌─ Home
  texture: TEXTURE
  chosenImg: URL ⤔ TEXTURE
  hDDN: seq URL
  hMainPanel, hLoginPanel, hDialogPanel: PANEL
  homePanelState: ℙPANEL ⤔ VISIBILITY
  isIntractable: ISINTRACTABLE
|
  hLoginPanel = {(InLogin, inField), (InPwrd, inField), (BtnLogin, btn)}
  hMainPanel = {(BtnUploadNewImg, btn), (BtnLoadByRegion, btn), (BtnDelDdnImg, btn),
(DdnHome, ddn), (BtnLlogOut, btn), (BtnUnitSize, btn), (BtnRegions, btn), (BtnExportCSV,
btn)}
  hDDN ∊ DdnHome
  ∃ url ∊ ran hDDN • texture = chosenImg url
  dom homeCurState = {hMainPanel, hLoginPanel}
  homePanelState = {(hMainPanel, SHOW ), (hLoginPanel, HIDE)} ∨ {(hMainPanel, HIDE
), (hLoginPanel, SHOW)}
└─

/* The login panel is visible in case the user is not authenticated, otherwise it is hidden */

┌─ hLoginPanelON
  Home
|
  homePanelState = {(hMainPanel, HIDE ), (hLoginPanel, SHOW)}
└─

┌─ hLoginPanelOFF
  Home
|
  homePanelState = {(hMainPanel, SHOW ), (hLoginPanel, HIDE)}
└─

┌─ hLoginPanelSwitchLaw
  AppData
  Home
|
  (loggedIn ∧ hLoginPanelOFF) ∨ (¬loggedIn ∧ hLoginPanelON)
└─

┌─ hToggleLoginPanel
  ΔHome
  evt?: EVENTS
|
  [(evt? =authSuccessEvt) ∧ hLoginPanelON′] ∨ [(evt? = authFailEvt) ∧ hLoginPanelOFF
′]
└─
```

/* At initialization, assume that user is not authenticated, show login panel and query auth state */

/* hInitEvt (home init signal) is equivalent of: qCurImgAndUrlEvt, qUnitEvt, qImgUrlsEvt, qAuthStateEvt */
/* to initialize Home we need: current working image, ddn-urls, unit-existence-query, auth-state
/* BtnRegions: without a unit, we will not measure */

```
┌─ HomeInit
  ΔSignal
  Home′
  hLoginPanelON′
|
  FiredEvt′ {hInitEvt}
  texture′ = Ø
  chosenImg′ = Ø
  hDDN′ = Ø
  isIntractible′ = {BtnLogin↦enable, BtnUploadNewImg↦enable, BtnDelDdnImg↦enable,
    BtnLoadByRegion↦enable,DdnHome↦disable, BtnLlogOut↦enable,
BtnUnitSize↦enable,
    BtnRegions↦enable, BtnExportCSV↦enable}
└─
```

/* Login process: the Home and AppProvider are involved in the business */

/* Example why to Z-document software: */
When the login process is to take place, the field of password is not empty
InPwrd ≠ Ø, when in: hLogIn

```
┌─ hUserLogIn
  ΔSignal
  Home
  getText: UITEXT
  clickedOn?: btn
  username!, password!: TEXT
|
  InPwrd ≠ Ø
  InLogin ≠ Ø
  clickedOn? = BtnLogin
  username! = getText InLogin
  password! = getText InPwrd
  FiredEvt′ {reqLoginEvt}
└─

┌─ AppLogin
  Signal
  AppProvider
  evt?: EVENTS
|
  evt? = reqLoginEvt
  [doLogin(evt?) ∧ FiredEvt(authSuccessEvt)] ∨ [¬doLogin(evt?) ∧ FiredEvt(authFailEvt)]
└─
```

/* at the start of Home panel, ddn may or may not have photos, if it has, the first one is auto-selected. If not, waiting for user to upload a photo. Btns of Unit, Region and CSV are disabled */

current working image, ddn-urls
/* informally:
when @home: => enable BtnUnitSize */

```
┌─ Home_BtnRegions_Enabling
  Home
  e?: EVENTS
  unitMagnitude?: ℤ
|
  unitMagnitude? > 0
  e? = unitUpdateEvt
  isIntractible′ = isIntractible ⊕ {BtnRegions↦enable}
└─
```

/* all possible hDDN events:

1. @Init: new seq of urls, or none:
- ddnUpdateEvt (has seqUrls, url: equal to first one, texture of the url)
2. ddn selection changed in home, texture update is expected
- reqTextureByUrlEvt (provide url in request)
- response: - ddnUpdateEvt (has url set to reqUrl, texture, ddn list is null)
3. user uploaded new img, url @App level not found in seqURLs, so an evt with new seqURL with
added new img's url and texture comes back, totally updating ddn, selecting the url in it and
updating texture:
exactly same as #1
overall: if seq ddn is null, from the existing ddn select the evt.url if found, then rest

```
texture
if url is not in the current-existing seq ddn, raise error

event exchange:
1.
req: hInitEvt
res: hDdnUpdateEvt
abt: res may have or have not url-texture for selection, in app side, lastReqUrl will =
head req hDDN
if, befure texture of head req hDDn obtained and sent back, usr req another url from list,
that one
sits in lastReqUrl and requested the texture of, from server, so the texture which will be
sent to
user, will always be the last one that user requested.

2.
req: reqTextureByUrlEvt
res: lastReqUrl will be updated by url? of req evt, provider messaged with url? as param.

3.
event: provider received texture of a url
if app's lastReqUrl is that, it is sent back to user, else, texture is discarded
so:
if lastReqUrl = providerUrl? then appFire(hDdnUpdateEvt) with hDDN, url, texture.

4.
req:
*/




/* @Init: hInitEvt sent to app, this is response of app*/
┌─ Home_DdnInit
  ΔHome
  e?: EVENTS
  ddnList?: seq URL
  texture?: TEXTURE
|
  e? = hDdnUpdateEvt
  texture’ = texture?
  hDDN’ = ddnList?
  url’ = head ddnList?
└─

/* @Selection REQ url? of user-selected being sent to app*/
┌─ HomeDdn_SelectionChanged
  ΔSignal
  ΔHome
  url!: URL
  getSelected: UITEXT
  e!: EVENTS
|
  url! = getSelected hDDN
  e! =
  FiredEvt′ {e!}
└─

/*
A response to:
- user selected a url of ddn, or
- uploaded a new image,
*/
/* @Selection RES */
┌─ Home_ImgUpdate
  ΔHome
  e?: EVENTS
  url?: URL
  texture?: TEXTURE
  getSelected: UITEXT
|
  url? ∈ ran hDDN
  e? = hDdnUpdateEvt
  getSelected′ hDDN = url?
  texture′ = texture?
└─

/* error case: raise error, using hDialogPanel of Home schema, defered!
/* left for reader exercise! */
/* @Err */
┌─ Home_ImgUpdate
  ΞHome
  e?: EVENTS
  url?: URL
  texture?: TEXTURE
  getSelected: UITEXT
|
  url? ∉ ran hDDN
  e? = hDdnUpdateEvt
└─

/* @App */
/* @hInitEvt Signal */
send down ddn, even if null, but req JS by proxy, for latest 10Urls
┌─ AppHomeInitEvtRes
```

```
  ΞAuthenticated
  ΞAppData
  ΔAppProvider
  e!, e?: EVENTS
  hDDN!: seq URL
|
  e? = hInitEvt
  hDDN! = hDDN
  if hDDN=Ø then dom asyncIntentQueue′ = dom asyncIntentQueue ∪ {getLatestUrls}
└─

/* @ */
┌─ AppGetUrlTexture
  ΞAuthenticated
  ΞAppData
  ΔAppProvider
  url?: URL
|
  url? ≠ Ø
  wellformed(url?)
  asyncIntentQueue′ = dom asyncIntentQueue ∪ {getTextureOfUrl}
└─
lastReqUrl

reqTextureByUrlEvt

hDdnUpdateEvt
  e?: EVENTS
  ddnList?: seq URL
  texture?: TEXTURE

    url! = getSelected hDDN
  e! =

==========================
┌─ AppProvider
  doLogin: EVENTS ⇸ BOOL
  asyncIntentQueue: ASYNQUEUE
└─
┌─ AppProviderInit
  AppProvider′
|
  doLogin′ = Ø
  asyncIntentQueue′ = Ø
└─

┌─ AppImgUrl
```

```
  curImg: Img
  urls: seq URL
└─

┌─ AppImgUrlInit
  AppImgUrlʼ
|
  curImgʼ = Ø
  urlsʼ = Ø
└─

┌─ AppUnit
  unitLatestʼ: MVector
  unitImgʼ: Img
└─

┌─ AppUnitInit
  AppUnitʼ
|
  unitLatestʼ = Ø
  unitImgʼ = Ø
└─

┌─ AppData
  loggedIn: BOOL
  scene: SCENE
  hDDN: seq URL
└─

┌─ Authenticated
  AppData
|
  loggedIn = T
└─
```

e! with url! of ddn goes to appProvider, chng AppImg downloading from innet, returning texture update
req-e!
=> AppImg void (bye appProvider) for its internal texture and set img new url
=> provider to download img from innet itself
=> after download update appImg texture and issue event of appTextureUpdate
=> home makes sure coming url in ddn, then select it (if it's not, adds it), and updates texture

in Unit scene:
- if user deleted a unit which was loaded,
or opened an img but didn't create any unit yet => isIntractible BtnAccept = disable

if cur img and url not available, the fist of ddn will be chosen, if no ddn url, then wait for upload
when img upload, the url returns back to app, if home receives "just uploaded", if ddn is empty,
will be placed there, else, add to ddn as last img.

logout btn must work like a master reset, as the next user is not necessarily the who was logged in
logout btn sends a reqNullReset to app, and app sends a reqSceneReset (any scene listens it)

/* hDDN changed issuing evt, if appImg.url is the same and appImg.texture not null, that will be sent as response, as "imgTextureUpdate", but if either texture is null or url different,
provider will obtain that img, then both appImg updated and event toward
cur-img-texture-update
fired
when a new-url comes in, sets to get its img download
when new url-img available, ddn of app inspected, if not contains, an image of
curImgTextureAndDdnUpdate fired
curDdnSelectionTextureUpdateEvt: will update ddn, select the url, and updates texture all together.
*/

/*
current working image is function of selection from ddn.

ddn on every update will receives a copy of app url-seq, and totally updates to it

on every full ddn update, it issues query event asking cur app img and texture and gets updated to

if ddn has no img or has any, and one uploaded, the url is added to ddn then selected.
if ddn has img and one is selected, the working image updates with it
Application code guarantees that any img app is working on, belongs to imgUrlsList
*/

```
/*
this, "program logic design using ISOx", is a "tool", that is all. That is how to understand
it. An
artist involved in sculpture art will not expect that one of his or her tools will do
everything,
but may use various tools, what is described in this book is "one of the tools". Chances
are
that if you put it to use "properly", you come to the same conclusion as I came to: it
really makes
the job easier, by preventing the logical errors from the begining. The cost of such errors
down the
road will be much higher. You may find out that it will help to reduce the redundancy
greatly, if not
to at all prevent it. These are what I have confidence it, after practicing this skill.
*/
```

```
── section RegionFSM parents standard_toolkit
└─

/* Specifying the concept of Region FSM */


── [ URL, TEXTURE, RNAME, GAMEOBJECT, VOID ] └─

/* Boolean definition and meaning */
── theorem d_HasIntegerType
  ⊢? d : ℤ
└─
──
  F == (d ∊ ℤ ∧ d ∉ ℤ)
└─
──
  T ==  (d ∊ ℤ ∨  d ∉ ℤ)
└─
──
  BOOL ::= T | F
└─
──
  DDNRNAME == -- ∪ RNAME
└─


──
  MSET == {A1 , A2 , B1 , B2 , C1 , C2 , D1 , D2}
└─
──
  MEASUREMENT == MSET ∪ {--}
└─
──
  STATE ::= CreateRegion | RegionUD | SegmentCUD
└─

┌─ Angles
  E1, E2: ℕ
|
  E1≥0
  E2≥0
└─

/* the image is loaded in the 3rd section of XY coordinate system */
┌─ POINT
  x,y: ℤ
|
  x < 0
  y < 0
└─

┌─ SEGMENT
  p1,p2: POINT
|
  p1 ≠ p2
└─

┌─ Region
  rname: RNAME
  diagonal: SEGMENT
  angles: Angles
  mSet: MSET ⤔ SEGMENT
└─

──
  RenderFrom == Region ⇸ VOID
└─
──
  SegmentOffIn_sUI == MSET ⇸ VOID
└─
──
  RegionDraftUpdate == POINT × POINT ⇸ VOID
└─
──
  BlockAndPromptSaveRegionDraft == RNAME ⇸ VOID
└─
──
  InitSegmentUpdatedSet = (MSET × BOOL) ⟶ {sName:MSET • sName ↦ F}
└─




/*
  rDDN: seq RNAME // region dropdown
  sDDN: seq MEASUREMENT// segment dropdown
  rDict: RNAME ⇸ Region // dictionary of region name to region data structure
  state: STATE // environment state, one of creationR | R-update/delete | segmentCRUD
  rDDNVal: ℕ // selected value in region dropdown
  sDDNVal: ℕ // selected value in segment dropdown
  img: TEXTURE // texture chosen in home
  url: URL // url of the chosen texture
  unit: ℕ // unit created in unit scene
  isRegionUpdated: BOOL // know if region-under-work is manipulated (and saving /
descarding as next ops)
  previousSegment: MEASUREMENT // so that if any segment of prev. segment is
```

```
rendered off, render on back,
  when newSegment chosen in sDDN
*/

┌─ RegionEnv
  rDDN: seq RNAME
  rDDNVal: ℕ

  sDDN: seq MEASUREMENT
  sDDNVal: ℕ

  rDict: RNAME ⇸ Region

  regionDraft: Region

  state: STATE

  img: TEXTURE
  url: URL
  unit: ℕ

  isRegionUpdated: BOOL
  isSegmentUpdated: MSET ⟶ BOOL
  previousSegment: MEASUREMENT
|
  rDDNVal ∈ dom rDDN
  sDDNVal ∈ dom sDDN
  unit > 0
  img ≠ Ø
  url ≠ Ø
  rand rDDN = dom rDict
  isSegmentUpdated ≠ Ø
└─

┌─ RegionEnv_Init
  RegionEnv'
  img?: IMG
  url?: URL
  unit?: UNIT
  rseq?: seq REGION
|
  img' = img?
  url' = url?
  unit' = unit?
  rDDN' = 1↦-- ⁀ {r:rseq? | (second r).rname ≠ -- • (first r+1) ↦ (second r).rname}
  rDict' = {r:rseq? • r.rname ↦ r}
  sDDN' = {1↦"---", 2↦A1, 3↦A2, 4↦B1, 5↦B2, 6↦C1, 7↦C2, 8↦D1, 9↦D2}
  rDDNVal′ = 1 ∧ rDDN′(rDDNVal′) = --
  sDDNVal′ = 1 ∧ sDDN′(sDDNVal′) = --

  regionDraft = Ø

  isRegionUpdated′ = F
  isSegmentUpdated = InitSegmentUpdatedSet(isSegmentUpdated)
  previousSegment′ = --

  state′ = CreateRegion
└─

/*

The process of creating new region donw by rFSM is defered here, as a result of creation, a new
region DS is sent to server for save and if saving succeeded the Region comes back, making the next op
to take place, changing state from CreateRegion to RegionUD, and getting selected to rDDN, while sDDN remains unselected
(it selects first value, i.e. --)

roles:
- rDict: latest region data structure from server, source of update region to original
- regionDraft: contains a copy of region from rDict to hold updates, and maybe saved to server ro discarded
- RegionUI: datastructure to render Region UI, copies from rDict
- SegmentUI: data structure to render Segment UI, copies from rDict

Only source of information for Region that updates all others, is rDict, every other related DS is passive, but
rDict is active DS.
*/

┌─ NewRegion_fromServer
  ΔRegionEnv
  r?: Region
|
  rDDN′ = rDDN ∪ {(#rDDN+1)↦r?.rname}
  rDDNVal′ = #rDDN+1
  regionDraft = r?
  rDict′ = rDict ∪ {r?.rname ↦ r?}

  sDDNVal′ = 1 ∧ sDDN′(sDDNVal′) = --

  isRegionUpdated = isRegionUpdated′ = F
  isSegmentUpdated = InitSegmentUpdatedSet(isSegmentUpdated)
  previousSegment = previousSegment′ = --

  state = CreateRegion
  state′ = RegionUD
└─

┌─ RegionUD_Entry
  ΔRegionEnv
|
  state′ = RegionUD
  state ≠ state′

  let region_name = rDDN′(rDDNVal′)
  ∃RenderFrom(rDict(region_name))
└─


┌─ RegionUD_to_CreateRegion
  ΔRegionEnv
|
  rDDNVal′ = 1 ∧ rDDN′(rDDNVal′) = --
  sDDNVal′ = 1 ∧ sDDN′(sDDNVal′) = --

  isRegionUpdated′ = F
  previousSegment′ = --

  state = RegionUD
  state′ = CreateRegion
└─

┌─ RegionUD_to_SegmentCUD
  ΔRegionEnv
|
  isRegionUpdated = isRegionUpdated′ = F

  previousSegment = --
  previousSegment′ = sDDN′(sDDNVal′)
  sDDN′(sDDNVal′) ≠ --
  rDDN′(rDDNVal′) ≠ --

  let region_name = rDDN′(rDDNVal′)
  let segment_name = sDDN′(sDDNVal′)
  ∃SegmentOffIn_sUI(region_name, segment_name)
  ∃MarkersAndFSM_to(rDict, region_name, segment_name)

  state = RegionUD
  state' = SegmentCUD
└─

┌─ SegmentCUD_Segment_SelectionChanged
  ΔRegionEnv
|
  sDDN'(sDDNVal') ≠ --
  rDDN'(rDDNVal') ≠ --

  previousSegment ≠ --
  let region_name = rDDN(rDDNVal)
  ∃RenderSegment(previousSegment, rDict, "on")


  previousSegment' = sDDN'(sDDNVal')
  ∃RenderSegment(previousSegment', rDict, "off")
  ∃MarkersAndFSM_to(rDict, region_name, previousSegment')

  state = state' = SegmentCUD
└─

┌─ SegmentCUD_Region_SelectionChange
  ΔRegionEnv
  p1?, p2?: POINT
  v?: VOID
|
  isRegionUpdated' = T
  rDDN'(rDDNVal') ≠ --
  ∃RegionDraftUpdate(p1?, p2?)
  let region_name = rDDN(rDDNVal)
  ∃BlockAndPromptSaveRegionDraft(region_name)
└─

┌─ SegmentCUD_ChgSaveServer
  ΔRegionEnv
  v?: VOID
  promptSaveAccepted?: BOOL
  p!: REPORT
|
  isRegionUpdated = T
  rDDN(rDDNVal) ≠ --
  promptSaveAccepted? = T
  let region_name = rDDN(rDDNVal)
  ∃ServerSaveRegionDraft(region_name)
  ∃BlockScene(v?)
  p! = PleaseWaitServerSavingGoinOn
└─
```

```
┌─ SegmentCUD_ChgDiscard
  ΔRegionEnv
  v?: VOID
  promptSaveAccepted?: BOOL
|
  isRegionUpdated = T
  isRegionUpdated' = F
  promptSaveAccepted? = F
  let region_name = rDDN(rDDNVal)
  ∃ResetDraft(region_name, rDict)
  ∃ResetSegmentUI(region_name, rDict)
  ∃UnBlockScene(v?)
└─

┌─ SegmentCUD_to_CreateRegion
  ΔRegionEnv
|
  rDDN'(rDDNVal') = --
  isRegionUpdated = F;

  state = SegmentCUD
  state' = CreateRegion
└─

┌─ SegmentCUD_to_RegionUD
  ΔRegionEnv
|
  rDDN'(rDDNVal') ≠ --
  isRegionUpdated = F;
  sDDN' ≠ sDDN

  state = SegmentCUD
  state' = RegionUD
└─
```

```
— section Leaf Measurement parents standard_toolkit
└─

— [ URI, PHOTO, ITEMID, USERNAME, PASSWORD, UNIX_EPOCH, EVENT ] └─
—
  LOGINEVT == EVENT ⤀ {TRUE, FALSE}
└─
—
  FIREEVT == EVENT ⤀ {QueryLoginState, ReqLogin}
└─
—
  DATE == UNIX_EPOCH
└─
—
  URL == URI
└─
—
  REQ == PHOTOS | REGINAMES
└─
—
  EVENT ::= IsLoggedIn | IsLoggedOut
└─
—
  TRUE == 1
└─
—
  FALSE == 0
└─
—
  BOOLEAN == {TRUE, FALSE}
└─
—
  SAVED == BOOLEAN
└─
—
  SEGMENT_NAME  == { A1, A2, B1, B2, C1, C2, D1, D2, E1, E2}
└─
—
  SEGMENT_SIZE == ℕ
└─
—
  POINT == X x Y
└─
—
  COORDINATES == POINT x POINT
└─
—
```

```
  REGIONALS == COORDINATES
└─
──
  SEGMENT == COORDINATES
└─
──
 OVERLAP == (REGIONALS x REGIONALS) ⟶ BOOLEAN
└─
──
  INSIDEOF == ( SEGMENT x REGIONALS) ⟶ BOOLEAN
└─

── section Main Data Structures parents standard_toolkit └─

┌─ UnitVectorDS
  magnitude: SEGMENT_SIZE
  url: URL
└─
┌─ RegionDS
  sgValue: SEGMENT_NAME ⟶ SEGMENT
  ofMagnitude: SEGMENT_NAME ⟶ SEGMENT_SIZE
  region: REGIONALS
  regionId: ITEMID
  url: URL
|
  ∀ s: sgValue • (ran(s) INSIDEOF region) = TRUE
└─
──
  ⊢? ∀ r1, r2: RegionDS | r1 ≠ r2 ∧ r1.url = r2.url • r1.region OVERLAP r2.region = TRUE
└─
┌─ DB
    munit: DATE ⤔ UnitVectorDS
    regions: ITEMID ⤔ RegionDS
    login: (USERNAME x PASSWORD) ⟶ BOOLEAN
└─

── section Application Classes parents standard_toolkit └─

┌─ class ProjectWideData
    ┌──
      PhotoURL: URL
      IsLoggedIn: BOOLEAN
    └─
    ┌─ Init
      PhotoURL′ = Ø
      IsLoggedIn′ = FALSE
    └─
    ┌─ OnQueryLoginState
      e!, e?: EVENT
     |
      e? ∊ dom FIREEVT
      EVENTFIRE e? = QueryLoginState
      e! ∊ dom LOGINEVT
      LOGINEVT e! = IsLoggedIn
    └─
└─
┌─ class CoverPanel
    ↾ ( Visibility , Username, Password )
    ┌──
      visibility: BOOLEAN
      username: USERNAME
      password: PASSWORD
    └─
    ┌─ Init
      visibility = TRUE
      username = Ø
      password = Ø
    └─
└─

┌─ class HomeSceneCtrl
    ↾ ( CoverPanelSet , BtnLogin_OnClick, UploadPhoto_OnClick , QueryByRegion_OnClick,
              RemovePhoto_OnClick, UnitScene_OnClick, RegionScene_OnClick )
    ┌──
      CoverPanel
    └─
    ┌─ Init
      e!: EVENT
    |
      e! ∊ dom EVENTFIRE
      QueryLoginState = EVENTFIRE e!
      CoverPanel'.Init
    └─
    ┌─ CoverPanelSet
      Δ ( CoverPanelSet )
      e?: EVENT
     |
      e? ∊ dom LOGINEVT
      CoverPanel.visibility' = LOGINEVT e?
    └─
    ┌─ RequestLogin
      e! ∊ dom FIREEVT
      EVENTFIRE e! = ReqLogin
      e!.data.username = CoverPanel.username
      e!.data.password = CoverPanel.password
    └─
└─
┌─ class HomeServiceProvider
    ↾ ( SetLoginState  )
    [ OutboundService ]
    ReqLoginService == OutboundService.LoginService
     ┌─ RequestLogin
      e? ∊ dom FIREEVT
      EVENTFIRE e? = ReqLogin
      ReqLoginService(e!.data.username, e!.data.password)
     └─
└─
/*
1. JS when logged in, must req srv of latest 10 photos by date (assumption: latest photos is
what user is interested to work with

2. user can make req of nav to next 10 photos from UI ddn related btns, and ddn will get updated

3. if a photo-url from ddn is chosen, but the photo can't be obtained from srv, del from ddn too

4. dialogue of Lynda pass to better learn UI

5. photo full CRUD, before going any other part. Warn user if to del photo, all related regions get del

6. all needed CRUDs:
- photo
- unit
- region
- measurements
*/
```