

Price Property Checker of ESELSystem1 for the ESEL System

Kangfeng Ye

April 27, 2016

1 Header

section *ESELHeader* **parents** *circus_toolkit*

This section gives all basic definitions that will be used in all three *Circus* models. And gateway related definitions are only used in the ESEL System 2.

First of all, three constants are defined. *MAX_ESEL* and *MAX_PID* stand for maximum number of displays and maximum number of product categories (or, products for short) in the system separately. And constant *MAX_GATEWAY* stands for maximum number of gateways.

| *MAX_ESEL* : \mathbb{N}
| *MAX_PID* : \mathbb{N}

Then all displays and products are identified by a tag plus a unique number which are defined in the free types *ESID* and *PID* below where the constructors *ES* and *PD* are the tags for displays and products. For an instance, number ten of the display is given *ES* 10 or *ES*(10). Similarly, *GID* gives all identities for gateways.

ESID ::= *ES*⟨1 .. *MAX_ESEL*⟩
PID ::= *PD*⟨1 .. *MAX_PID*⟩

The type of product price is defined as an abbreviation to natural numbers \mathbb{N} .

Price == \mathbb{N}

The unit response is defined as a free type with two constants: *uok* and *ufail*.

UStatus ::= *uok* | *ufail*

The response from this program to the environment is a set of product identities of which the price is not updated successfully due to 1) no linked ESEL ID to the product or 2) failed update to its linked ESEL. The first reason is given the status constant *NA* and the second is provided the constructor *fail*⟨*ESID*⟩.

FStatus ::= *fail*⟨*ESID*⟩ | *NA*

Two channels are provided to update the map from ESEL ID to product ID. *updateallmap* will clear all stored map and use the input map as new map, while *updatemap* just updates a partial map. In this map, one ESEL can be linked to up to one product. However, one product may associate with multiple ESELs.

channel *updateallmap* : *ESID* \leftrightarrow *PID*
channel *updatemap* : *ESID* \leftrightarrow *PID*

Similarly, two channels are provided to update the price information. *updateallprice* will clear all price information and use the input price information as new price, while *updateprice* just updates price partially.

```
channel updateallprice : PID  $\rightarrow$  Price
channel updateprice : PID  $\rightarrow$  Price
```

The *update* channel gives a signal to the program to start update process.

```
channel update
```

The *failures* channel returns all failed products and related error reasons after update. Since one product may associate with multiple displays, the return status is a power set of *FStatus* to denote which specific displays that the product links are updated unsuccessfully. But it is worth noting that *NA* and *fail* must not occur in a product's return set at the same time because they can not be both no associate display and associate display update fail.

```
channel failures : PID  $\rightarrow$  P FStatus
```

The internal *resp* event is used to collect update responses from all displays and *terminate* event is for completing the collection.

```
channel resp : PID  $\times$  FStatus
channel terminate
channelset RespInterface == { resp, terminate }
```

This *uupdate* event is to update one ESEL to the specific price, and *ures* for update response from this ESEL. And *udisplay* is used to synchronise the show of price on all ESELs at the same time and *finishdisplay* is used to wait for display completion of all ESELs. That is the similar case for *uinit* and *ufinishinit* that are for initialisation synchronisation.

```
channel uupdate : ESID  $\times$  Price
channel ures : ESID  $\times$  UStatus
channel uinit, finishuinit
channel udisplay, finishudisplay
```

And *display* is used to synchronise the show of price on all gateways (or ESELs) at the same time and *finishdisplay* is used to wait for display completion of all gateways (or ESELs). That is the similar case for *init* and *finishinit* that are for initialisation synchronisation.

```
channel init, finishinit
channel display, finishdisplay
```

The channels below are for communication between the ESEL system and displays. The *write* event writes price to a display, and the *read* event reads price from the display. *ondisplay* turns on the related display and *offdisplay* turns off it conversely.

```
channel write : ESID  $\times$  Price
channel read : ESID  $\times$  Price
channel ondisplay : ESID
channel offdisplay : ESID
```

2 Price Checker

section *ESELPriceChecker* **parents** *ESELHeader*

```

process PriceChecker  $\hat{=}$  begin
  state State == [ dummy : {0} ]
  Init == [ (State)' | dummy' = 0 ]
  AOnDisplay  $\hat{=}$  eid : ESID • ondisplay.eid →
    (| | | e : (ESID \ {eid}) || ∅ || • (offdisplay.e → Skip))
  AOffDisplay  $\hat{=}$  (| | | e : ESID || ∅ || • (offdisplay.e → Skip))
  ACheckMap  $\hat{=}$  p : Price ; eid : ESID ; pid : PID •
    (updateallmap.({eid ↦ pid}) → updateallprice.({pid ↦ p}) →
    update → write.eid.p → (
      (read.eid.p → AOnDisplay(eid) ; failures.({}) → Skip)
      □ (read.eid?x : (x ≠ p) → AOffDisplay ;
        failures.({pid ↦ {(fail eid)}}) → Skip)
    )
  )
  ACheckNoMap  $\hat{=}$  p : Price ; eid : ESID ; pid : PID •
    (updateallmap.({}) → updateallprice.({pid ↦ p}) → update →
    AOffDisplay ; failures.({pid ↦ {NA}}) → Skip)
  ACheck  $\hat{=}$  var p : Price ; eid : ESID ; pid : PID •
    ACheckMap(p, eid, pid) □ ACheckNoMap(p, eid, pid)
  • Init ; AOffDisplay ; (μX • (ACheck) ; X)
end

```

```

channelset ESELSysntemInterface == { | updateallprice, updateprice,
  updatemap, updateallmap, update, ondisplay, offdisplay,
  write, read, failures | }

```

section *ESELSystem1Checker* **parents** *ESELPriceChecker*, *ESELSystem1*

```

process ESELSystem1Checker  $\hat{=}$ 
  (PriceChecker || ESELSysntemInterface || ESELSystem1)

```