Circus Solution of ESELSystem2 for the ESEL System

Kangfeng Ye

April 27, 2016

1 Header

$\textbf{section}\ \ ESELHeader\ \textbf{parents}\ \ circus\_toolkit$

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.

$$\begin{array}{|l} MAX\_ESEL : \mathbb{N} \\ MAX\_PID : \mathbb{N} \end{array}$$

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.

$$\begin{array}{l} ESID ::= ES \langle\!\langle 1 \mathrel{..} MAX\_ESEL \rangle\!\rangle \\ PID ::= PD \langle\!\langle 1 \mathrel{..} MAX\_PID \rangle\!\rangle \end{array}$$

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 $ufail$.

$$UStatus ::= uok \mid ufail$$

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 $fail \langle\!\langle ESID \rangle\!\rangle$.

$$FStatus ::= fail \langle\!\langle ESID \rangle\!\rangle \mid NA$$

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.

$$\begin{array}{l} \textbf{channel}\ \ updateallmap : ESID \pfun PID \\ \textbf{channel}\ \ updatemap : ESID \pfun PID \end{array}$$

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.

$$\begin{array}{l}
\mathbf{channel}\ \ updateallprice : PID \pfun Price \\
\mathbf{channel}\ \ updateprice : PID \pfun Price
\end{array}$$

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

$$\mathbf{channel}\ \ update$$

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.

$$\mathbf{channel}\ \ failures : PID \pfun \mathbf{P}\ FStatus$$

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

$$\begin{array}{l}
\mathbf{channel}\ \ resp : PID \times FStatus \\
\mathbf{channel}\ \ terminate \\
\mathbf{channelset}\ \ RespInterface == \{\!|\ resp, terminate\ |\!\}
\end{array}$$

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.

$$\begin{array}{l}
\mathbf{channel}\ \ uupdate : ESID \times Price \\
\mathbf{channel}\ \ ures : ESID \times UStatus \\
\mathbf{channel}\ \ uinit, finishuinit \\
\mathbf{channel}\ \ udisplay, finishudisplay
\end{array}$$

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.

$$\begin{array}{l}
\mathbf{channel}\ \ init, finishinit \\
\mathbf{channel}\ \ display, finishdisplay
\end{array}$$

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.

$$\begin{array}{l}
\mathbf{channel}\ \ write : ESID \times Price \\
\mathbf{channel}\ \ read : ESID \times Price \\
\mathbf{channel}\ \ ondisplay : ESID \\
\mathbf{channel}\ \ offdisplay : ESID
\end{array}$$

2 ESEL System 2

$$\textbf{section}\ \ ESELSystem2\ \textbf{parents}\ \ ESELHeader$$

$$MAX\_GATEWAY : \mathbb{N}$$

$$GID ::= GW \langle\!\langle 1 \mathrel{..} MAX\_GATEWAY \rangle\!\rangle$$

The map from ESELs to gateways, *gwmap*, is defined as a total function. One ESEL is linked to up to one gateway. However, one gateway may associate with multiple ESELs.

$$\begin{array}{|l}
gwmap : ESID \rightarrow GID \\
\hline
gwmap = \{(ES\,1, GW\,1), (ES\,2, GW\,1), (ES\,3, GW\,2)\}
\end{array}$$

The channels below are used to communicate between the server and gateways, or between gateway internals. The server uses *gupdateprice* to send price information with ESEL IDs to the corresponding gateway, while *gfailure* is used to get back the udpate result from the gateway.

$$\begin{array}{l}
\textbf{channel}\ \ gupdateprice : GID \times (ESID \pfun Price) \\
\textbf{channel}\ \ gfailure : GID \times \mathbf{P}\ ESID
\end{array}$$

gresp and *gterminate* are used in the internal of gateways to collection update results from each ESEL and terminate after collection.

$$\begin{array}{l}
\textbf{channel}\ \ gresp : ESID \\
\textbf{channel}\ \ gterminate \\
\textbf{channelset}\ \ GRespInterface == \{\!|\ gresp, gterminate\ |\!\}
\end{array}$$

ESEL Server Process The process for overall control of the system, named *ESELServer*, is defined as an explicitly defined process.

$$\textbf{process}\ \ ESELServer \mathrel{\widehat{=}} \textbf{begin}$$

The *ESELServer* has three state components: *pumap* for mapping from ESELs to products, *ppmap* for mapping from products to their price, and *response* for the response of one update to the environment.

$$\begin{array}{l}
\textbf{state}\ \ State == [\, pumap : ESID \pfun PID\, ;\, ppmap : PID \pfun Price; \\
\qquad response : PID \pfun (\mathbf{P}\ FStatus)\,]
\end{array}$$

Initially, these three state components all are empty.

$$Init == [\, (State)' \mid pumap' = \varnothing \land ppmap' = \varnothing \land response' = \varnothing\,]$$

The *UpdateMap* schema updates part of the ESELs to products map according to the input map, while the *UpdateAllMap* schema discards all map and uses new input map as *pumap*.

$$\begin{array}{l}
UpdateMap == [\, \Delta State\, ;\, map? : ESID \pfun PID \mid \\
\qquad pumap' = pumap \oplus map? \land ppmap' = ppmap \land response' = response\,] \\
UpdateAllMap == [\, \Delta State\, ;\, map? : ESID \pfun PID \mid \\
\qquad pumap' = map? \land ppmap' = ppmap \land response' = response\,]
\end{array}$$

The *NewPrice* updates part of price information stored, while the *AllNewPrice* discards all price information stored and uses input price as *ppmap*.

$$\begin{aligned}
&\mathit{NewPrice} == [\,\Delta \mathit{State}\,;\, \mathit{price}? : \mathit{PID} \text{ ⇸ } \mathit{Price} \mid \\
&\qquad \mathit{ppmap}' = \mathit{ppmap} \oplus \mathit{price}? \land \mathit{pumap}' = \mathit{pumap} \land \mathit{response}' = \mathit{response}\,] \\
&\mathit{AllNewPrice} == [\,\Delta \mathit{State}\,;\, \mathit{price}? : \mathit{PID} \text{ ⇸ } \mathit{Price} \mid \\
&\qquad \mathit{ppmap}' = \mathit{price}? \land \mathit{pumap}' = \mathit{pumap} \land \mathit{response}' = \mathit{response}\,]
\end{aligned}$$

AUpdatemap is an action defined to update ESELs to products map: either partial update by *updatemap* event or complete update by *updateallmap* event.

$$\begin{aligned}
&\mathit{AUpdatemap} \mathrel{\widehat{=}} \mathit{updatemap}?\mathit{map} \rightarrow \big(\mathit{UpdateMap}\big) \\
&\qquad \Box\ \mathit{updateallmap}?\mathit{map} \rightarrow \big(\mathit{UpdateAllMap}\big)
\end{aligned}$$

Similarly, *ANewPrice* is an action defined to update products to price map: either partial update by *updateprice* event or complete update by *updateallprice* event.

$$\begin{aligned}
&\mathit{ANewPrice} \mathrel{\widehat{=}} \mathit{updateprice}?\mathit{price} \rightarrow \big(\mathit{NewPrice}\big) \\
&\qquad \Box\ \mathit{updateallprice}?\mathit{price} \rightarrow \big(\mathit{AllNewPrice}\big)
\end{aligned}$$

If the update to an ESEL fails, *AUpdateUnitFail* sends the failure by *resp* to the response collection action *CollectResp*.

$$\mathit{AUpdateUnitFail} \mathrel{\widehat{=}} \mathit{eid} : \mathit{ESID} \bullet \mathit{resp}.(\mathit{pumap}(\mathit{eid})).(\mathit{fail}\ \mathit{eid}) \rightarrow \mathbf{Skip}$$

Or if the product has not been allocated the corresponding ESELs, it sends back a response to state this error *NA*. The behaviour is defined in the *AUpdateNoUnit* action.

$$\mathit{AUpdateNoUnit} \mathrel{\widehat{=}} \mathit{pid} : \mathit{PID} \bullet \mathit{resp}.\mathit{pid}.\mathit{NA} \rightarrow \mathbf{Skip}$$

For all products without associate ESELs, they send the failures independently.

$$\begin{aligned}
&\mathit{ARespNoUnit} \mathrel{\widehat{=}} \big|\big|\big|\ \mathit{pid} : (\operatorname{dom} \mathit{ppmap} \setminus \operatorname{ran} \mathit{pumap}) \llbracket\, \varnothing \rrbracket \bullet \\
&\qquad \mathit{AUpdateNoUnit}(\mathit{pid})
\end{aligned}$$

For each gateway, *AUpdateGateways* sends all price for the ESELs which are linked to the gateway and gets back update result. Then for each failure, the action passes it to *AUpdateUnitFail*, and finally writes to *response*.

$$\begin{aligned}
&\mathit{AUpdateGateway} \mathrel{\widehat{=}} \mathit{gid} : \mathit{GID} \bullet \\
&\qquad \mathit{gupdateprice}.\mathit{gid}!((\operatorname{dom}(\mathit{gwmap} \rhd \{\mathit{gid}\})) \lhd (\mathit{pumap} \fatsemi \mathit{ppmap})) \rightarrow \\
&\qquad \mathit{gfailure}.\mathit{gid}?\mathit{uids} \rightarrow (\big|\big|\big|\ \mathit{uid} : \mathit{uids} \llbracket\, \varnothing \rrbracket \bullet \mathit{AUpdateUnitFail}(\mathit{uid}))
\end{aligned}$$

Update of price to ESELs is an interleave of *AUpdateGateway* for all gateways.

$$\mathit{AUpdateGateways} \mathrel{\widehat{=}} \big|\big|\big|\ \mathit{gid} : \mathit{GID} \llbracket\, \varnothing \rrbracket \bullet \mathit{AUpdateGateway}(\mathit{gid})$$

Then the update of all products, given in the action *AUpdateProducts*, is the interleave of the update of price to ESELs through gateways and the action for the case without associate ESELs. Then it follows a *terminate* event to finish the update.

$$\begin{aligned}
&\mathit{AUpdateProducts} \mathrel{\widehat{=}} (\mathit{AUpdateGateways} \llbracket\, \varnothing \mid \varnothing\, \rrbracket \mathit{ARespNoUnit}); \\
&\qquad \mathit{terminate} \rightarrow \mathbf{Skip}
\end{aligned}$$

$$\begin{array}{l} AddOneFailure == [\,\Delta State\ ;\ pid? : PID\ ;\ fst? : FStatus \mid \\ \quad (pid? \in \mathrm{dom}\ response \Rightarrow \\ \qquad response' = response \oplus \{pid? \mapsto (response(pid?) \cup \{fst?\})\}) \land \\ \quad (pid? \notin \mathrm{dom}\ response \Rightarrow \\ \qquad response' = response \cup \{pid? \mapsto \{fst?\}\}) \land \\ \quad ppmap' = ppmap \land pumap' = pumap\,] \end{array}$$

The *CollectResp* action is to collect responses from all units and write them into the *response* variable. It recursively waits for the response from the units, or terminates if required.

$$\begin{array}{l} ACollectResp \mathrel{\widehat=} \mu X \bullet \\ \quad ((resp?pid?fst \to \big(AddOneFailure\big)\ ;\ X) \mathrel{\Box} terminate \to \mathbf{Skip}) \end{array}$$

Then update of all products and response collection behaviours are put together into *AUpdateResp* action. It is a parallel composition of *AUpdateProducts* and *CollectResp* actions and they are synchronised with *resp* and *terminate* events. Finally, these internal events are hidden.

$$\begin{array}{l} AUpdateResp \mathrel{\widehat=} \\ \quad (AUpdateProducts \llbracket \varnothing \mid RespInterface \mid \{response\} \rrbracket ACollectResp) \\ \quad \big\backslash RespInterface \end{array}$$

The overall price update action is given in *AUpdatePrice*, which accepts a *update* event from its environment, then clears *response*, updates the price, sends *display* event to make all ESELs show their price at the same time, then feeds back the response to the environment.

$$\begin{array}{l} AUpdatePrice \mathrel{\widehat=} update \to response := \varnothing; \\ \quad AUpdateResp\ ;\ display \to finishdisplay \to failures.response \to \mathbf{Skip} \end{array}$$

The overall behaviour of the *ESELServer* process is given by its main action. It initializes at first, then repeatedly provides ESEL map update, price map, or price update to its environment.

$$\begin{array}{l} \bullet \big(Init\big)\ ;\ init \to finishinit \to \mathbf{Skip}; \\ \quad (\mu X \bullet (AUpdatemap \mathrel{\Box} ANewPrice \mathrel{\Box} AUpdatePrice)\ ;\ X) \end{array}$$

end

Gateway Process The *Gateway* process is defined as parametrised process.

$$\mathbf{process}\ Gateway \mathrel{\widehat=} gid : GID \bullet \mathbf{begin}$$

It has two state components: *pumap* for the map from ESELs to price, and *failed* for a set of ESELs which update unsuccessfully.

$$\mathbf{state}\ State == [\,pumap : ESID \pfun Price\ ;\ failed : \mathbf{P}\ ESID\,]$$

Initially, both are empty.

$$Init == [\,(State)' \mid pumap' = \varnothing \land failed' = \varnothing\,]$$

The map only can be updated completely and can not be updated partially.

$$\begin{array}{l} \mathit{UpdateAllMap} == [\, \Delta \mathit{State} \,;\, \mathit{map?} : \mathit{ESID} \pfun \mathit{Price} \mid \\ \quad \mathit{pumap}' = \mathit{map?} \land \mathit{failed}' = \mathit{failed} \,] \end{array}$$

The map is updated after input from *ESELServer* through the *gupdateprice* channel.

$$\mathit{AUpdateallmap} \mathrel{\widehat=} \mathit{gupdateprice}.\mathit{gid}?\mathit{map} \rightarrow \big(\mathit{UpdateAllMap} \big)$$

A parameterised action, *AUpdateUnitPrice*, is given to update the price (specified by the formal *pid* parameter) to an ESEL (given by the formal *uid* parameter). It sends the price to the specified ESEL by the *uupdate* event, and then waits for the response from the ESEL. If the return status is not successful (*ufail*), it sends the result to response collection action *CollectResp* below, then terminates. Otherwise, it terminates immediately.

$$\begin{array}{l} \mathit{AUpdateUnitPrice} \mathrel{\widehat=} \mathit{uid} : \mathit{ESID} \bullet \\ \quad \mathit{uupdate}.\mathit{uid}.(\mathit{pumap}\ \mathit{uid}) \rightarrow \mathit{ures}.\mathit{uid}?\mathit{rst} \rightarrow \\ \quad (\big(\mathit{rst} = \mathit{ufail}\big) \mathbin{\&} \mathit{gresp}!\mathit{uid} \rightarrow \mathbf{Skip} \\ \quad \Box \ \big(\mathit{rst} = \mathit{uok}\big) \mathbin{\&} \mathbf{Skip}) \end{array}$$

Updates of all ESELs in this gateway are put in an iterated interleave, then follow a *gterminate* event to finish the updates.

$$\begin{array}{l} \mathit{AUpdateAllUnits} \mathrel{\widehat=} ((\big|\big|\big|\ \mathit{eid} : (\mathrm{dom}\ \mathit{pumap}) \ [\![\, \varnothing \,]\!] \bullet \mathit{AUpdateUnitPrice}(\mathit{eid})) \\ \quad ; \mathit{gterminate} \rightarrow \mathbf{Skip}) \end{array}$$

The *CollectResp* action is to collect responses from all units and write them into the *response* variable. It recursively waits for the response from the units, or terminates if required.

$$\begin{array}{l} \mathit{AGCollectResp} \mathrel{\widehat=} \mu X \bullet \\ \quad ((\mathit{gresp}?\mathit{uid} \rightarrow \mathit{failed} := \mathit{failed} \cup \{\mathit{uid}\} \,;\, X) \ \Box\ \mathit{gterminate} \rightarrow \mathbf{Skip}) \end{array}$$

Then update of all products and response collection behaviours are put together into *AUpdateResp* action. It is a parallel composition of *AUpdateProducts* and *CollectResp* actions and they are synchronised with *resp* and *terminate* events. Finally, these internal events are hidden.

$$\begin{array}{l} \mathit{AGUpdateResp} \mathrel{\widehat=} \\ \quad (\mathit{AUpdateAllUnits} \ [\![\, \varnothing \mid \mathit{GRespInterface} \mid \{\mathit{failed}\} \,]\!] \ \mathit{AGCollectResp}) \\ \quad \big\backslash \mathit{GRespInterface} \end{array}$$

The overall price update action is given in *AUpdatePrice*, which accepts a *gupdateprice* event from its environment, then clears *failed*, updates the price, sends update results to the server, and waits for *display* event to make all ESELs in this gateway show their price at the same time.

$$\begin{array}{l} \mathit{AGUpdatePrice} \mathrel{\widehat=} \mathit{AUpdateallmap} \,;\, \mathit{failed} := \varnothing; \\ \quad \mathit{AGUpdateResp} \,;\, \mathit{gfailure}.\mathit{gid}!\mathit{failed} \rightarrow \mathit{display} \rightarrow \mathit{udisplay} \rightarrow \\ \quad \mathit{finishudisplay} \rightarrow \mathit{finishdisplay} \rightarrow \mathbf{Skip} \end{array}$$

The overall behaviour of the *Gateway* process is given by its main action. It initializes at first, then repeatedly provides ESEL map update, price map, or price update to its environment.

$$
\begin{aligned}
&\bullet \big(\mathit{Init} \big) \mathbin{;} \mathit{init} \rightarrow \mathit{uinit} \rightarrow \mathit{finishuinit} \rightarrow \mathit{finishinit} \rightarrow \mathbf{Skip};\\
&\quad (\mu X \bullet (\mathit{AGUpdatePrice}) \mathbin{;} X)
\end{aligned}
$$

$\mathbf{end}$

ESEL Process Each ESEL is defined as a parameterised process with the formal parameter—ESEL ID.

$$\mathbf{process}\ \mathit{ESEL2} \mathrel{\widehat=} \mathit{eid} : \mathit{ESID} \bullet \mathbf{begin}$$

The process has two state components: *price* for the price to display, and *status* for the status of ESEL.

$$\mathbf{state}\ \mathit{State} == [\, \mathit{price} : \mathit{Price} \mathbin{;} \mathit{status} : \mathit{UStatus} \,]$$

Initially, the price is equal to 0 and the status is *uok*.

$$\mathit{Init} == [\, (\mathit{State})' \mid \mathit{price}' = 0 \land \mathit{status}' = \mathit{uok} \,]$$

The *Update* action provides its environment (*Gateway*) the update of price for the associated product. It accepts the *uupdate* event with the price, then writes the price to *price*. After that, it writes the price to the display unit, and reads back the value to compare with the original price. If it is equal, it sends back status *uok* by the *ures* event. Otherwise, it sends back status *ufail*. Accordingly, *status* is updated.

$$
\begin{aligned}
&\mathit{Update} \mathrel{\widehat=} \mathit{uupdate.eid?x} \rightarrow \mathit{price} := x \mathbin{;} \mathit{write.eid.price} \rightarrow \mathit{read.eid?y}\\
&\quad \rightarrow ((y = \mathit{price}) \mathbin{\&} \mathit{ures.eid.uok} \rightarrow \mathit{status} := \mathit{uok}\\
&\qquad \Box\ (y \neq \mathit{price}) \mathbin{\&} \mathit{ures.eid.ufail} \rightarrow \mathit{status} := \mathit{ufail})
\end{aligned}
$$

The *Display* action accepts the *udisplay* event. If the status is *uok*, then the associated display is turned on. Otherwise, the display is turned off.

$$
\begin{aligned}
&\mathit{Display} \mathrel{\widehat=} \mathit{udisplay} \rightarrow (\\
&\qquad (\mathit{status} = \mathit{uok}) \mathbin{\&} \mathit{ondisplay.eid} \rightarrow \mathbf{Skip}\\
&\quad\ \Box\ (\mathit{status} = \mathit{ufail}) \mathbin{\&} \mathit{offdisplay.eid} \rightarrow \mathbf{Skip})\\
&\quad\ ; \mathit{finishudisplay} \rightarrow \mathbf{Skip}
\end{aligned}
$$

$$\mathit{NotUpdateDisplay} \mathrel{\widehat=} \mathit{udisplay} \rightarrow \mathit{offdisplay.eid} \rightarrow \mathit{finishudisplay} \rightarrow \mathbf{Skip}$$

The initial behaviour of the process is given in the action *AInit* which initialises the state at first, and then turns off the display.

$$\mathit{AInit} \mathrel{\widehat=} \big(\mathit{Init} \big) \mathbin{;} \mathit{uinit} \rightarrow \mathit{offdisplay.eid} \rightarrow \mathit{finishuinit} \rightarrow \mathbf{Skip}$$

The overall behaviour of the process is given by its main action. It specifies that after initialisation the process repeatedly provides update or display to its environment.

$$\bullet\ \mathit{AInit} \mathbin{;} (\mu X \bullet ((\mathit{Update} \mathbin{;} \mathit{Display}) \mathbin{\Box} \mathit{NotUpdateDisplay}) \mathbin{;} X)$$

end

System All ESELS which are registered with the same gateway synchronise on unit initialisation and display events.

$$\begin{array}{l}
\mathbf{channelset}\ \mathit{InterESELInterface2} == \{\!|\ \mathit{uinit}, \mathit{finishuinit}, \\
\qquad\qquad\qquad \mathit{udisplay}, \mathit{finishudisplay}\ |\!\} \\
\mathbf{process}\ \mathit{ESELS2} \mathrel{\widehat=} \mathit{gid} : \mathit{GID} \bullet \\
\quad (\Vert\ \mathit{eid} : (\mathrm{dom}\,(\mathit{gwmap} \rhd \{\mathit{gid}\})) \llbracket\ \mathit{InterESELInterface2} \rrbracket \bullet \mathit{ESEL2}(\mathit{eid}))
\end{array}$$

Each gateway is in parallel with its linked ESELs in the *GatewayESELS* process. And all gateays synchronise on gateway initialisation and display events which is defined as the *Gateways* process.

$$\begin{array}{l}
\mathbf{channelset}\ \mathit{InterGWInterface2} == \{\!|\ \mathit{init}, \mathit{finishinit}, \mathit{display}, \mathit{finishdisplay}\ |\!\} \\
\mathbf{channelset}\ \mathit{GWESELInterface2} == \{\!|\ \mathit{uinit}, \mathit{finishuinit}, \mathit{uupdate}, \mathit{ures}, \\
\qquad\qquad\qquad \mathit{udisplay}, \mathit{finishudisplay}\ |\!\} \\
\mathbf{process}\ \mathit{GatewayESELS} \mathrel{\widehat=} \mathit{gid} : \mathit{GID} \bullet \\
\quad (\mathit{Gateway}(\mathit{gid}) \llbracket\ \mathit{GWESELInterface2}\ \rrbracket\ \mathit{ESELS2}(\mathit{gid})) \setminus \mathit{GWESELInterface2} \\
\mathbf{process}\ \mathit{Gateways} \mathrel{\widehat=} \Vert\ \mathit{gid} : \mathit{GID} \llbracket\ \mathit{InterGWInterface2} \rrbracket \bullet \\
\quad \mathit{GatewayESELS}(\mathit{gid})
\end{array}$$

Finally, the ESEL System 2 is simply the parallel composition of the *ESELServer* and the *Gateways*, and communications between them are hidden.

$$\begin{array}{l}
\mathbf{channelset}\ \mathit{ServerGWInterface} == \{\!|\ \mathit{init}, \mathit{finishinit}, \mathit{gupdateprice}, \mathit{gfailure}, \\
\qquad\qquad\qquad \mathit{display}, \mathit{finishdisplay}\ |\!\} \\
\mathbf{process}\ \mathit{ESELSystem2} \mathrel{\widehat=} \\
\quad (\mathit{ESELServer} \llbracket\ \mathit{ServerGWInterface}\ \rrbracket\ \mathit{Gateways}) \setminus \mathit{ServerGWInterface}
\end{array}$$