Reactive and Event Based Systems
Assignment 2

Sofie Harning hsk270
Signe Scholer smg399

January 10, 2021

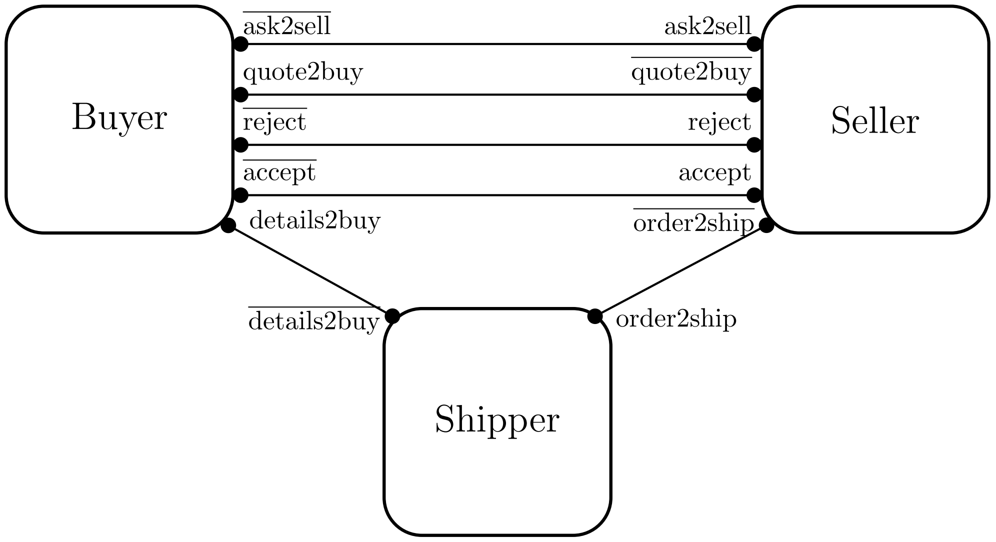
Buyer
Seller
Shipper
$\overline{\text{ask2sell}}$
ask2sell
quote2buy
$\overline{\text{quote2buy}}$
$\overline{\text{reject}}$
reject
$\overline{\text{accept}}$
accept
details2buy
$\overline{\text{order2ship}}$
$\overline{\text{details2buy}}$
order2ship

Figure 1: Caption

Part 1

1.1

See fig. 1.

1.2

$\text{BU} \overset{\text{def}}{=} \overline{\text{ask2sell}}.\text{quote2buy}.(\overline{\text{accept2sell}}.\text{details2buy}.0 + \overline{\text{reject2sell}}.0)$

$\text{BU}_1 \overset{\text{def}}{=} \text{quote2buy}.(\overline{\text{accept2sell}}.\text{details2buy}.0 + \overline{\text{reject2sell}}.0)$

$\text{BU}_2 \overset{\text{def}}{=} \overline{\text{accept2sell}}.\text{details2buy}.0$

$\text{BU}_3 \overset{\text{def}}{=} \text{details2buy}.0$

$\text{BU}_4 \overset{\text{def}}{=} \overline{\text{reject2sell}}.0$

$\text{SE} \overset{\text{def}}{=} \text{ask2sell}.\overline{\text{quote2buy}}.(\text{accept2sell}.\overline{\text{order2ship}}.0 + \text{reject2sell}.0)$

$\text{SE}_1 \overset{\text{def}}{=} \overline{\text{quote2buy}}.(\text{accept2sell}.\overline{\text{order2ship}}.0 + \text{reject2sell}.0)$

$\text{SE}_2 \overset{\text{def}}{=} \text{accept2sell}.\overline{\text{order2ship}}.0$

$\text{SE}_3 \overset{\text{def}}{=} \overline{\text{order2ship}}.0$

$\text{SE}_4 \overset{\text{def}}{=} \text{reject2sell}.0$

$\text{SH} \overset{\text{def}}{=} \text{order2ship}.\overline{\text{details2buy}}.0$

$\text{SH}_1 \overset{\text{def}}{=} \overline{\text{details2buy}}.0$

1.3

The relaxed Seller process can receive a reject or accept message without first having sent a quote. It can be used safely, as long as we do not change buyer, as the buyer will make sure it has received a quote before accepting or rejecting the seller.

1.4

$$
\begin{aligned}
\textit{Buyer} = {} & \overline{\text{ask2sellA}}(\text{"}\textit{chips}\text{"}).(\text{quoteA2buy(priceA)}.\overline{\text{ask2sellB}}(\text{"}\textit{chips}\text{"}).\text{quoteB2buy(priceB)}). \\
& \text{(if (priceA} < \text{20 and priceA} <= \text{priceB)} \\
& \text{then } \overline{\text{accept2sellA}}\text{.details2buy(invoice).0} \\
& \text{else if(priceB} < \text{20)} \\
& \text{then } \overline{\text{accept2sellB}}\text{.details2buy(invoice).0} \\
& \text{else } \overline{\text{reject2sell}}\text{.0}
\end{aligned}
$$

Part 2

2.1 Implementation of seller and shipper

We decided to have all interfaces in one file, and name all the interfaces consistently f.ex. `BuyerSellerInterface` is the interface from buyer to seller. This is also the case for our input- and outputPorts.

When the price is too high:

```
$ jolie SellerService.ol
Quoted buyer 22DKK for chips.
The price was rejected with messsage: Not ok to buy chips for 22

$ jolie ShipperService.ol
[No output. Does not terminate, since it is never contacted.]

$ jolie BuyerService.ol
Rejected price at 22DKK, since it's higher than 20DKK.
```

When the price is low enough:

```
$ jolie ShipperService.ol
Invoice sent to buyer: chips for 19DKK from Seller

$ jolie SellerService.ol
Quoted buyer 19DKK for chips.
The price was accepted with message: Ok to buy chips for 19

$ jolie BuyerService.ol
Accepted price at 19DKK, since it's lower than 20DKK.
Received the invoice from Shipper: You have bought chips for 19DKK from Seller
```

2.2 Implement a second seller

Our sellers are called `Seller` and `Seller0`. `SellerService0.ol` is a copy of `SellerService.ol`, except it runs on different ports.

2.3 Implement Extended Buyerservice

Both sellers can run with `BuyerServiceExtended.ol`.

*

One acceptable price:

```
$ jolie ShipperService.ol
Invoice sent to buyer: chips for 19DKK from Seller

$ jolie SellerService.ol
Quoted buyer 19DKK for chips.
The price was accepted with message: Ok to buy chips for 19

$ jolie SellerService0.ol
Quoted buyer 22DKK for chips.
The price was rejected with messsage: Not ok to buy chips for 22
```

```
$ jolie BuyerServiceExtended.ol
Accepted 19 from Seller, rejected 22 from Seller0.
Received the invoice from Shipper: You have bought chips for 19DKK from Seller
```

Two acceptable prices, accept the lowest:

```
$ jolie ShipperService.ol
Invoice sent to buyer: chips for 13DKK from Seller0

$ jolie SellerService.ol
Quoted buyer 19DKK for chips.
The price was rejected with messsage: Not ok to buy chips for 19

$ jolie SellerService0.ol
Quoted buyer 13DKK for chips.
The price was accepted with message: Ok to buy chips for 13

$ jolie BuyerServiceExtended.ol
Accepted 13 from Seller0, rejected 19 from Seller.
Received the invoice from Shipper: You have bought chips for 13DKK from Seller0
```

No acceptable prices:

```
$ jolie ShipperService.ol
[No output. Does not terminate, since it is never contacted.]

$ jolie SellerService.ol
Quoted buyer 21DKK for chips.
The price was rejected with messsage: Not ok to buy chips for 21

$ jolie SellerService0.ol
Quoted buyer 22DKK for chips.
The price was rejected with messsage: Not ok to buy chips for 22

$ jolie BuyerServiceExtended.ol
Rejected both 21 and 22 from Seller and Seller0, respectively.
```

Bonus

We have implemented a random seller service, which can be started in place of `SellerService.ol`, and works with both `BuyerService.ol` and `BuyerServiceExtended.ol`. The random seller chooses a price between 22 and 16 at random.

This is output from a run, where the price was accepted:

```
$ jolie SellerServiceRandom.ol
The price of chips is: 17
Quoted buyer 17DKK for chips.
The price was accepted with message: Ok to buy chips for 17
```

And output from a run, where the price was rejected:

```
$ jolie SellerServiceRandom.ol
The price of chips is: 22
```

```
Quoted buyer 22DKK for chips.
The price was rejected with messsage: Not ok to buy chips for 22
```