# Reactive and Event-based Systems

**Assignment 3: Choreographies in CCS and Jolie**

Thomas T. Hildebrandt

Software, Data, People & Society Section
Department of Computer Science
Copenhagen University, Denmark

## Part 1:

In Fig. 3, we show a BPMN choreography inspired from [4]. The choreography is based on a variant of the Buyer-Seller protocol described in [2] and involves three participants, a Buyer, a Seller and a Shipper. After asking the Seller for a quote and getting the reply, the Buyer may either Accept or Reject. If the Buyer accepts, the Seller sends an Order to the Shipper, which subsequently sends the detailed confirmation directly to the Buyer.

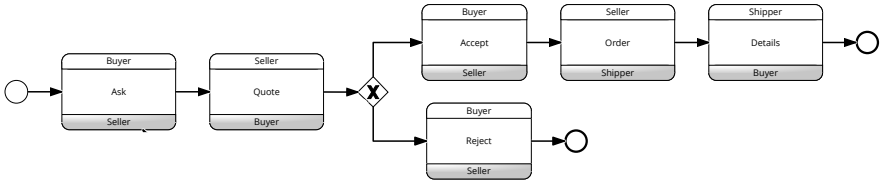


**Fig. 1.** BPMN Choreography for Buyer-Seller-Shipper example.

Below we modelled the Buyer, Seller and Shipper processes as value-passing CCS processes [1], making explicit that the Buyer asks for a price on chips and accepts if the price is lower than 20 and that the Seller always replies 17.

$$
\begin{aligned}
\mathit{Buyer} = \;& \overline{\texttt{ask2sell}}(\text{"}\mathit{chips}\text{"}).\texttt{quote2buy}(\texttt{price}).\\
& (\texttt{if}\,(\texttt{price} < 20)\,\texttt{then}\\
& \overline{\texttt{accept2sell}}(\text{"Ok to buy chips for "} + \texttt{price}).\texttt{details2buy}(\texttt{invoice}).0\\
& \texttt{else}\,\overline{\texttt{reject2sell}}(\text{"Not ok to buy chips for "} + \texttt{price}).0)\\
\mathit{Seller} = \;& \texttt{ask2sell}(\texttt{product}).\overline{\texttt{quote2buy}}(17).\\
& (\texttt{accept2sell}(\texttt{order}).\overline{\texttt{order2ship}}(\texttt{order}).0 +\\
& \texttt{reject2sell}(\texttt{order}).0)\\
\mathit{Shipper} = \;& \texttt{order2ship}(\texttt{product}).\overline{\texttt{details2buy}}(\text{"invoice for "} + \texttt{product}).0)
\end{aligned}
$$

The process uses six channels: `ask2sell`, `quote2buy`, `details2buy`, `accept2sell`, `reject2sell`and `order2ship`- one for each interaction in the choreography.

### Exercises

1.1 Draw the interface diagrams for Buyer, Seller and Shipper.

1.2 Draw the transition system for the process $(Buyer|Seller|Shipper)\backslash Channels$, where $Channels =$

$$\{\texttt{ask2sell}, \texttt{quote2buy}, \texttt{details2buy}, \texttt{accept2sell}, \texttt{reject2sell}, \texttt{order2ship}\}$$

(i.e. all channels are restricted). Name the intermediate states and write the process terms for each state.

1.3 Argue how the following relaxed Seller process is different from the original and if it can be used safely instead of the original in the choreography (if all channels are restricted):

$$\begin{aligned} Seller = \; & \texttt{ask2sell}(\texttt{product}).\overline{\texttt{quote2buy}}(17).0 \quad | \\ & \texttt{accept2sell}(\texttt{order}).\overline{\texttt{order2ship}}(\texttt{order}).0 \quad | \\ & \texttt{reject2sell}(\texttt{order}).0 \end{aligned}$$

1.4 Extend the CCS model such that the Buyer asks for quotes from two Sellers (Seller1 and Seller2) and accepts the lowest quote or rejects both. Try also to draw a choreography diagram for your extended process (abstracting away from the rule determining which quote gets accepted).

## Part 2:

Below is a model of the choreography as a DCR Graph with the hard-coded values (using the DCR Choreographies with Time and Data [3]. It can be found and simulated at `https://sim.dcrgraphs.net?code=6eeaaa70-f267-49cf-9849-5799d5709e1d`

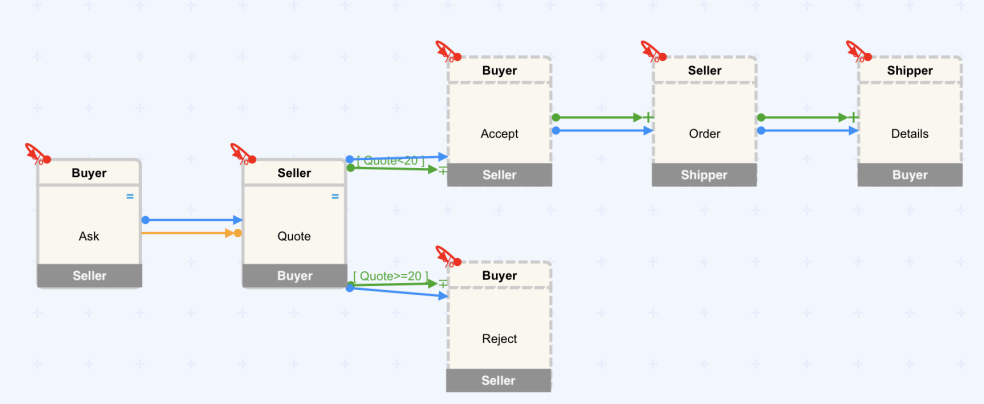


**Fig. 2.** DCR Choreography for Buyer-Seller-Shipper example.

And below is the DCR Graph for the Buyer end-point. It can be found and simulated at `https://sim.dcrgraphs.net?code=6eeaaa70-f267-49cf-9849-5799d5709e1d`

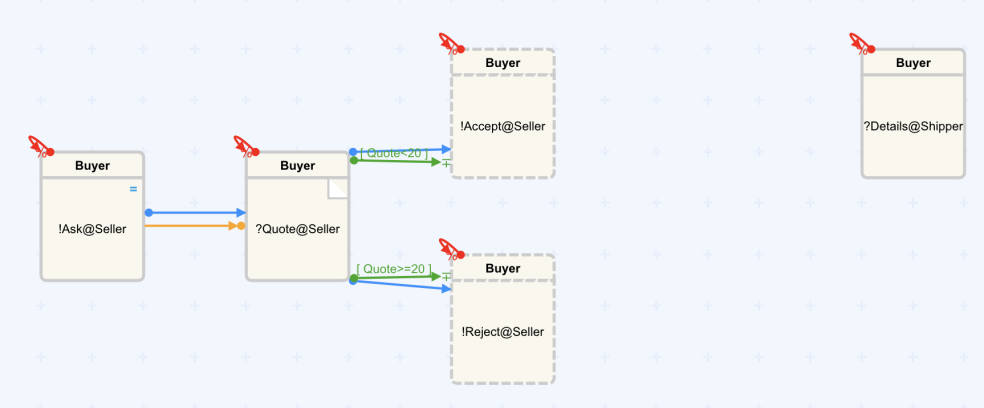


**Fig. 3.** End-point DCR Graph for Buyer

### Exercises

2.1 Model the end-point DCR Graphs for the Seller and Shipper. Argue why they look like they do and relate to the CCS processes. Optional: Try to model the extended Buyer modelled in 1.4 and the corresponding DCR choreography.

2.2 Implement the Buyer, Seller and Shipper end-points in a programming language of your choice (an example implementation of the Buyer in Jolie is given below). You may implemented the relaxed Seller mentioned in Exercises 1.3. You may also allow the seller and shipper to accept multiple concurrent requests.

2.3 Implement two Sellers, one that gives a too high price and one that gives a acceptable price. Make test runs and include prints of the result in the terminals.

2.4 Implement the extended Buyer that asks for quotes from two Sellers (Seller1 and Seller2) and accepts the lowest quote below 20 or rejects both. Test with your sellers.

Below is an implementation of the Buyer processes in Jolie 1.10.

```
 from SellerShipperServiceInterfaceModule import SellerInterface
from BuyerServiceInterfaceModule import BuyerShipperInterface, BuyerSellerInterface

include "console.iol"
service BuyerService {
```

```
execution{ single }

outputPort Seller {
     location: "socket://localhost:8000"
     protocol: http { format = "json" }
     interfaces: SellerInterface
}

inputPort ShipperBuyer {
     location: "socket://localhost:8001"
     protocol: http { format = "json" }
     interfaces: BuyerShipperInterface
}
inputPort SellerBuyer {
     location: "socket://localhost:8002"
     protocol: http { format = "json" }
     interfaces: BuyerSellerInterface
}
main {
    ask@Seller("chips")
        {[quote(price)]{
            if (price <20) {
                println@Console( "price lower than 20")()
                accept@Seller("Ok to buy chips for " + price)
                [details(invoice)]
                println@Console( "Received "+invoice+" from Shipper!")()}
            } else {
            println@Console( "price not lower than 20")()
            reject@Seller("Not ok to buy chips for " + price)
            }
        }
    }
}
```

using the interface module `BuyerServiceInterfaceModule`

```
interface BuyerShipperInterface {
    OneWay:
        details( string)
}

interface BuyerSellerInterface {
    OneWay:
        quote( int)
}
```

and the interface module `SellerShipperServiceInterfaceModule`

```
interface SellerInterface {
    OneWay:
ask( string ),
        accept( string ),
        reject( string )
}

interface ShipperInterface {
    OneWay:
        order( string )
}
```

## References


1. An introduction to milner's ccs (2005)
2. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centered programming for web services. ACM Trans. Program. Lang. Syst. 34(2), 8:1–8:78 (2012)
3. Hildebrandt, T.T., López, H.A., Slaats, T.: Declarative choreographies with time and data. In: Di Francescomarino, C., Burattin, A., Janiesch, C., Sadiq, S. (eds.) Business Process Management Forum. pp. 73–89. Springer Nature Switzerland, Cham (2023)
4. Hildebrandt, T.T., Slaats, T., López, H.A., Debois, S., Carbone, M.: Declarative choreographies and liveness. In: Formal Techniques for Distributed Objects, Components, and Systems. FORTE. Lecture Notes in Computer Science, vol. 11535. Springer, Cham. (2019)