Reactive and Event Based Systems: Part II

Choreographies in CCS and Jolie

December 20, 2021

Thomas Hildebrandt,
Software, Data, People & Society
Datalogisk Institut Københavns Universitet

KØBENHAVNS UNIVERSITET

Road map of todays exercsies

- 10:15-11: Choreographies
- 11:15-12: Assignment 2: Choreographies in valuepassing CCS and Jolie

Choreographies

Global model of the order of *interactions* between processes.

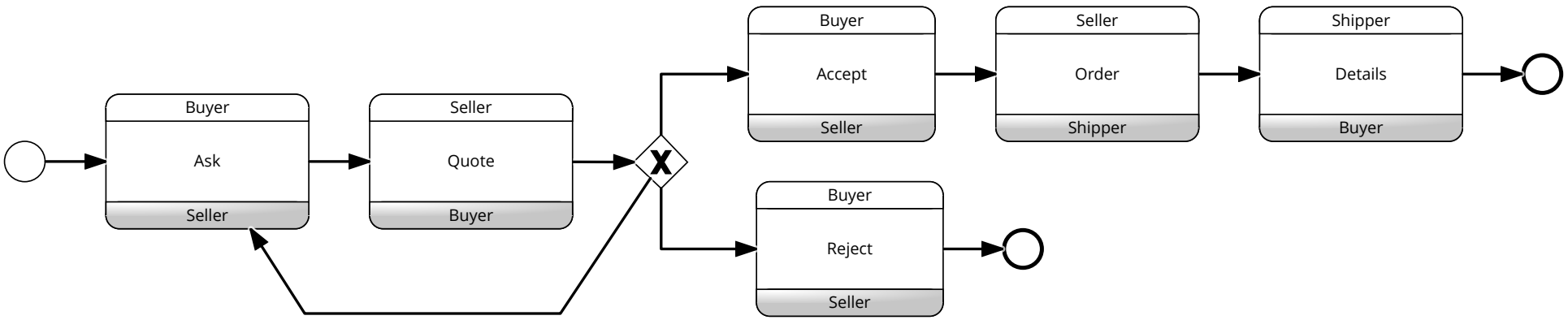


Fig. 1. BPMN Choreography for Buyer-Seller-Shipper example.

Many other industrial notations: Message Sequence Charts, UML Sequence Diagrams, WS-CDL

Many formal models (almost all imperative)

What does a choreography model give us ?

a simple global overview of a distributed system

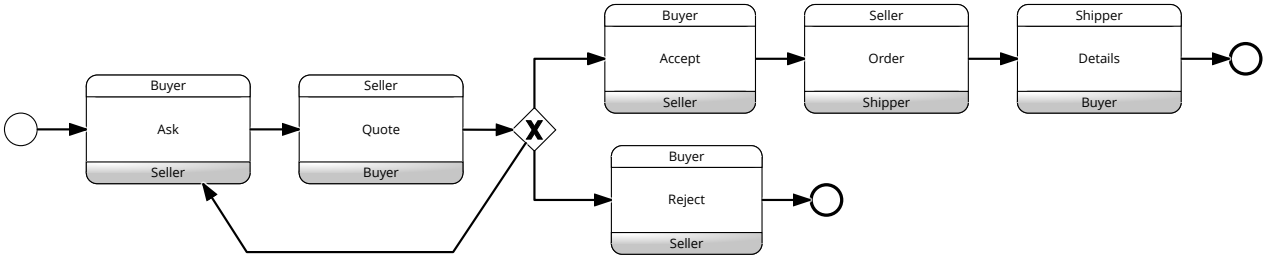
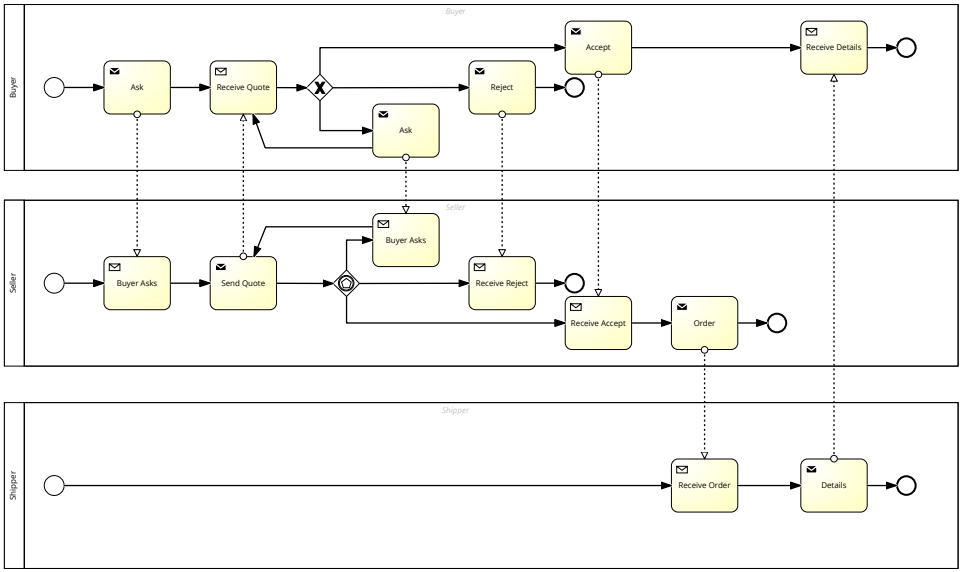


Fig. 1. BPMN Choreography for Buyer-Seller-Shipper example.

End-point projections



Guarantees for deadlock-freedom
-(but not liveness properties)

Liveness & Flexibility

- What if we wanted that the Buyer should eventually Accept or Reject ?
- What if we wanted to state, e.g. that the Buyer could ask any number of times before the seller gives a quote, but the seller should eventually give a quote?
- What if we wanted the Buyer to ask two or more sellers concurrently ?

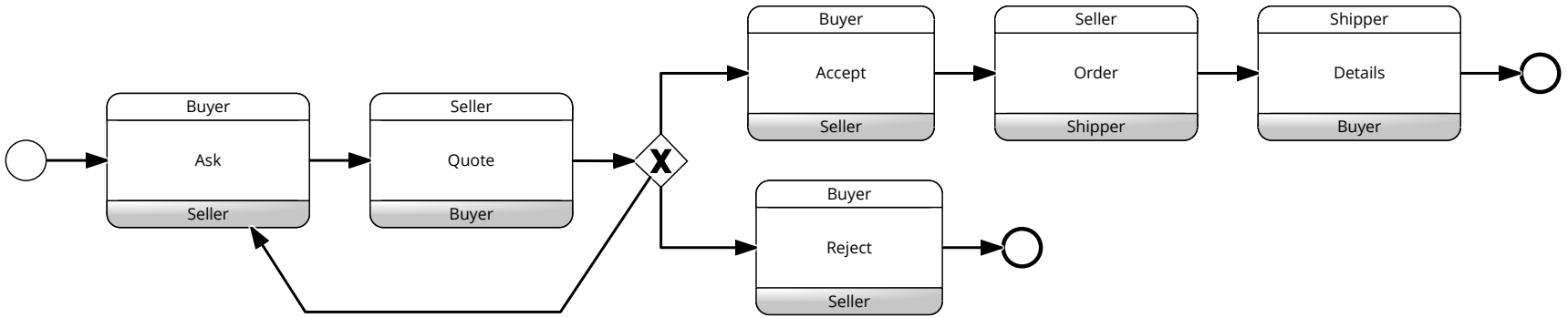
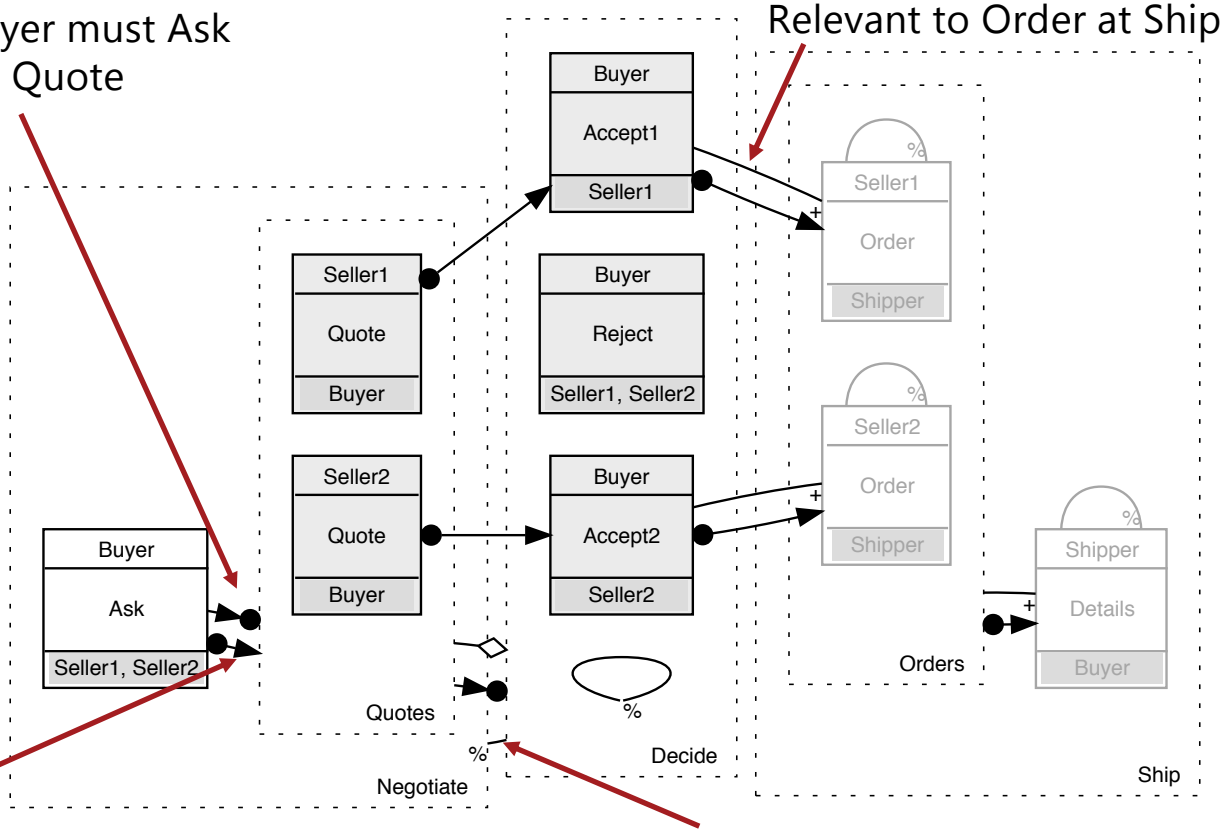


Fig. 1. BPMN Choreography for Buyer-Seller-Shipper example.

Declarative Choreographies as DCR Graph of Interactions

Condition: Buyer must Ask
Before Sellers Quote

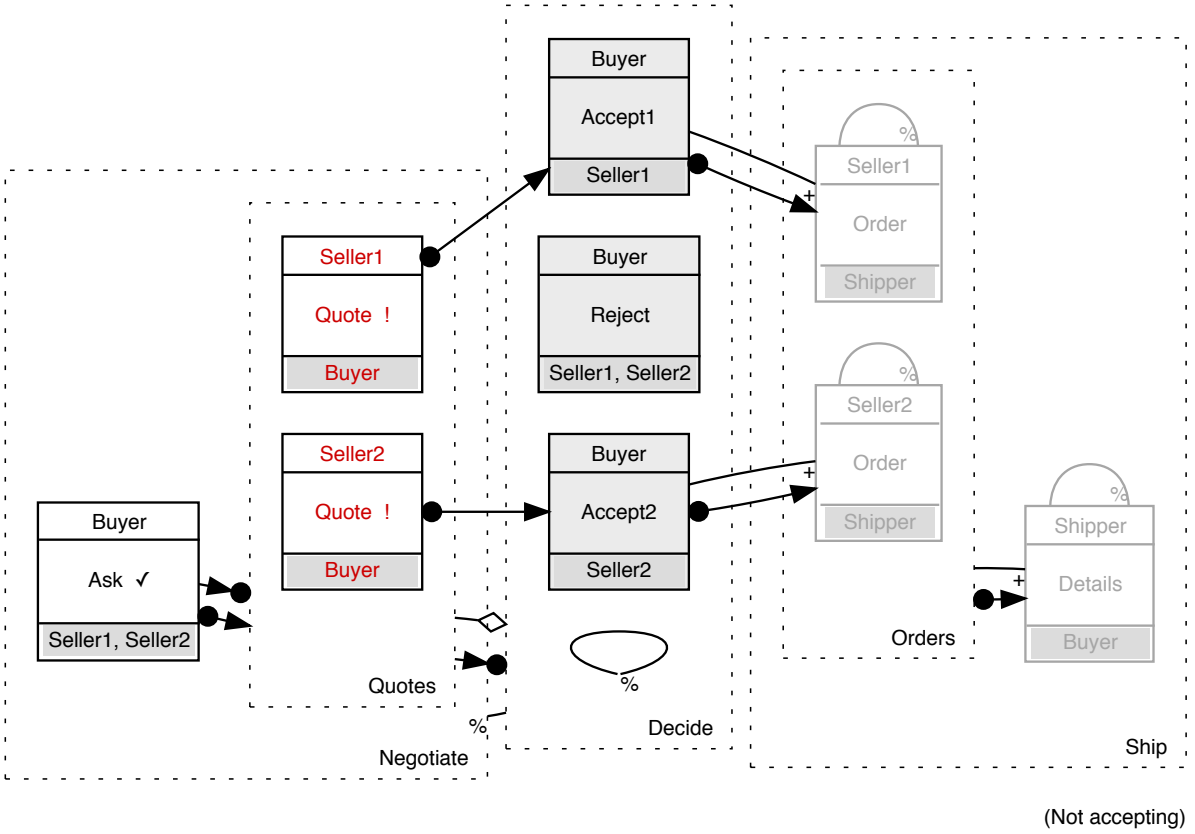
Inclusion: If Buyer Accepts it becomes
Relevant to Order at Shipper



Response: Sellers must eventually Quote if Buyer Asks

Exclusion: Once Buyer Accept or Reject,(Accepting)
Sellers can no longer Quote & Buyer can no longer Ask

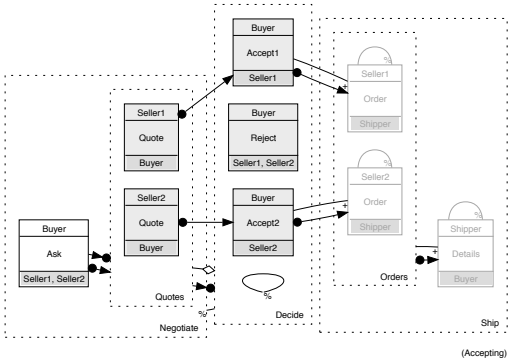
Marking of DCR Graph after executing Ask



Transition System Semantics of DCR Graphs

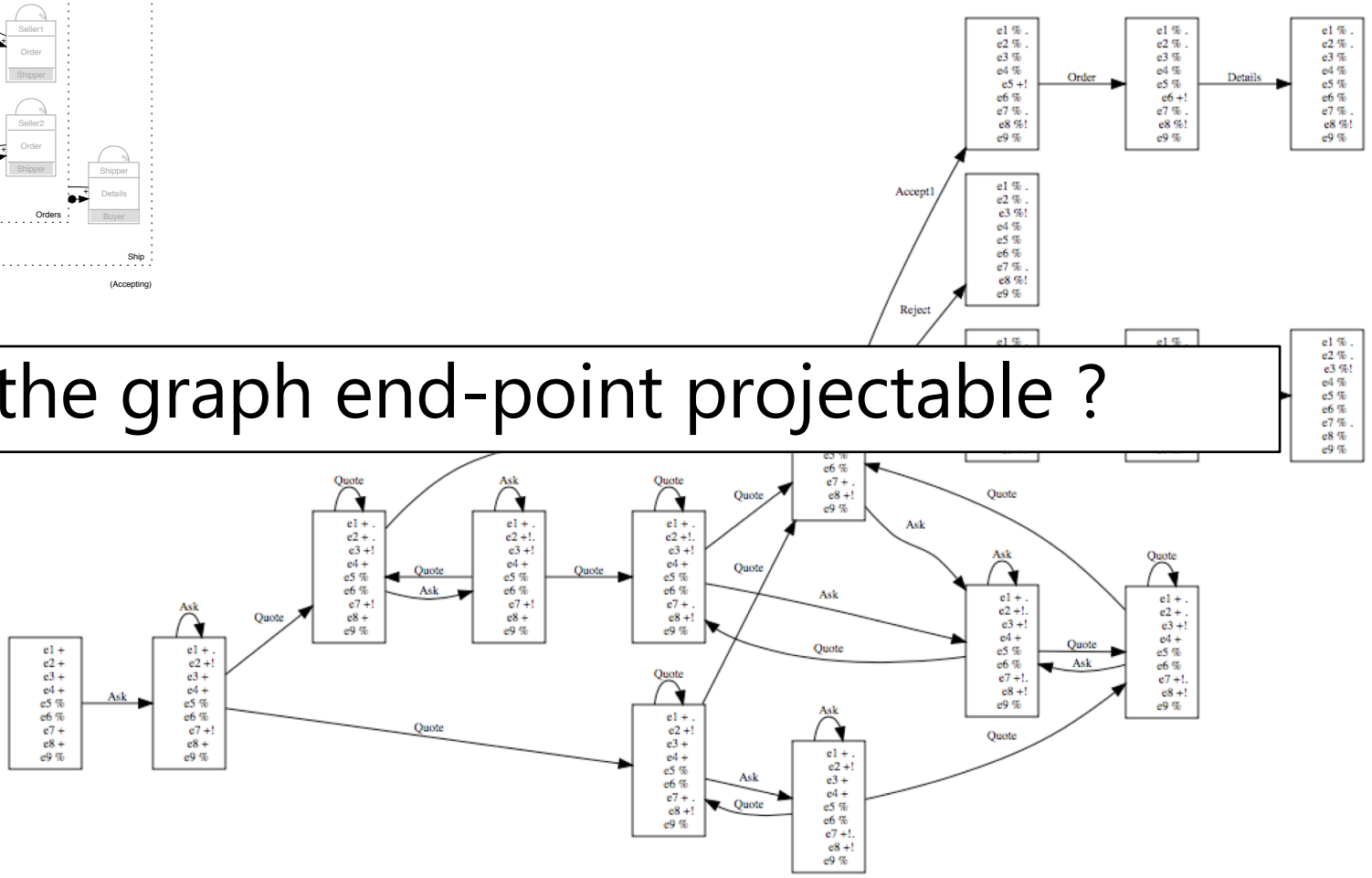
dcr.tools/forte19

G=



Is the graph end-point projectable ?

LETSR(G)=



Interactions & End-point projection of Interactions

Definition 1. *An* interaction *is a triple* $(a, r \to R)$*, in which the action* $a \in \mathsf{A}$ *is* initiated *by the role* r *and* received *by the roles* $R \subset_{\mathsf{fin}} \mathsf{R} \backslash \{r\}$*, i.e a finite set of roles distinct from* r*. Define* $\mathsf{Initiator}((a, r \to R)) = r$*. We use the shorthand* $(a, r \to r')$ *for interactions between two participants* $(a, r \to \{r'\})$*. We denote by* IA *the set of all interactions.*

Definition 2. *For an interaction* $\alpha = (a, r' \to R')$*, define the* end-point projection of α at r *by:*

$$\alpha_{|r} = \begin{cases} (a, r' \to r) & \text{when } r \in R' \\ (a, r' \to R') & \text{when } r' = r \\ \tau & \text{otherwise} \end{cases} \tag{1}$$

Causal criteria for end-point projectability

Definition 15. *Let* $G = (E, M, L, \ell, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\diamond, \rightarrow+, \rightarrow\%)$ *be a DCR graph and let* $e, e' \in E$ *be events of* E. *Then there is a* direct dependency $e' \preceq e$ *from* e' *to* e *iff either of the following conditions are true*

1. $e' = e$,
2. $e'(\rightarrow\bullet \cup \bullet\rightarrow \cup \rightarrow+ \cup \rightarrow\% \cup \rightarrow\diamond)e$,
3. $\exists e''.\ e'(\rightarrow+ \cup \rightarrow\%)e''(\rightarrow\bullet \cup \rightarrow\diamond)e$,
4. $\exists e''.\ e' \bullet\rightarrow e'' \rightarrow\diamond e$.

LOCAL PROPERTIES!

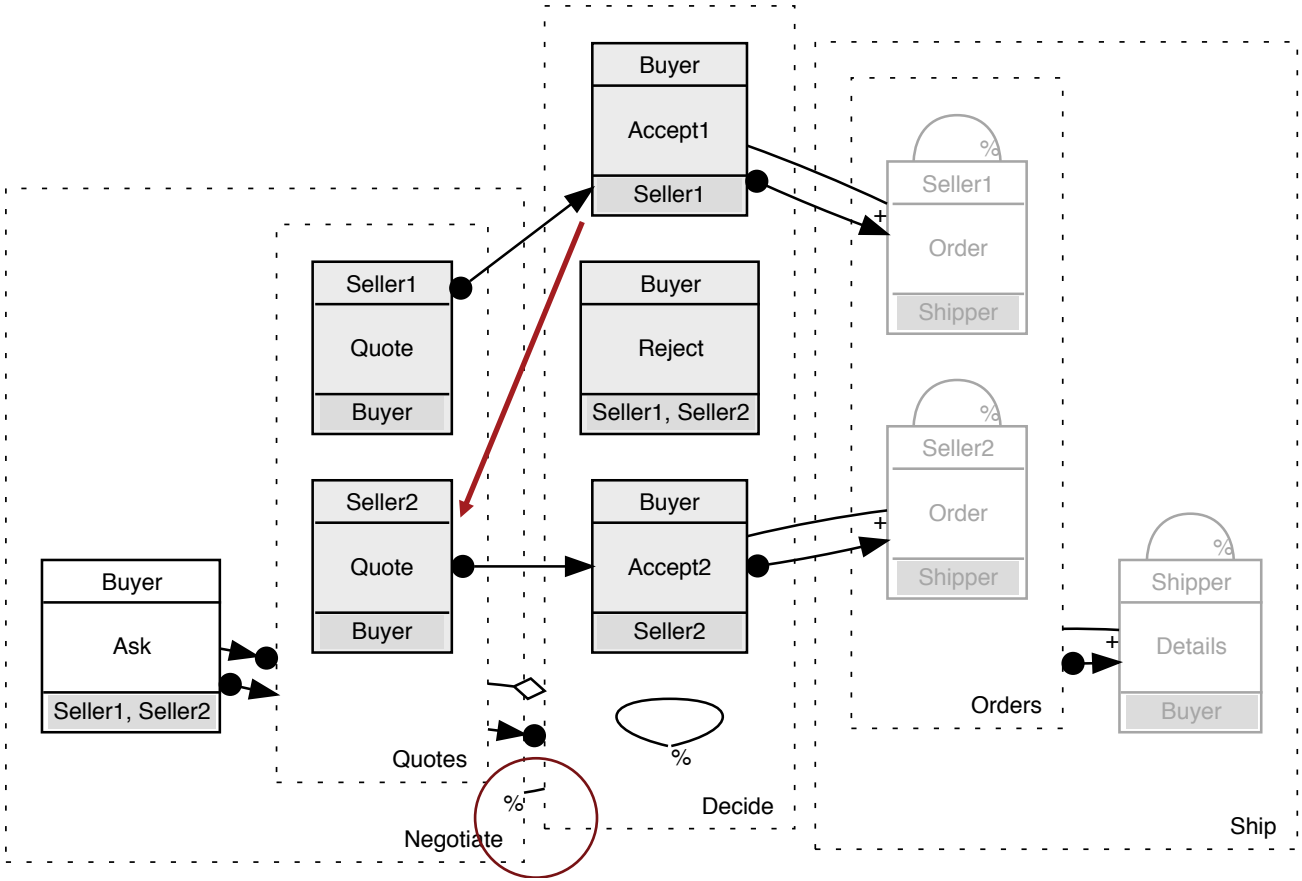
The Initiator of a Choreography Activity MUST have been involved (as Initiator or Receiver) in any *direct dependency* Choreography Activity.

Definition 17. *Let* (G, A, R) *be a DCR choreography and* ℓ *the labelling function of* G; *and let* $r \in R$ *be a role. This choreography is* end-point projectable *for* r *iff for all* e, *if* $\mathsf{Initiator}(e) = r$ *and* $e' \preceq e$, *then* $\ell(e')_{|r} \neq \tau$,

Computed in linear time !

End-point projectable ?

No – we have a direct dependency to an interaction initiated by Seller2 from an interaction not involving Seller2



Corrected (polite) end-point projectable version

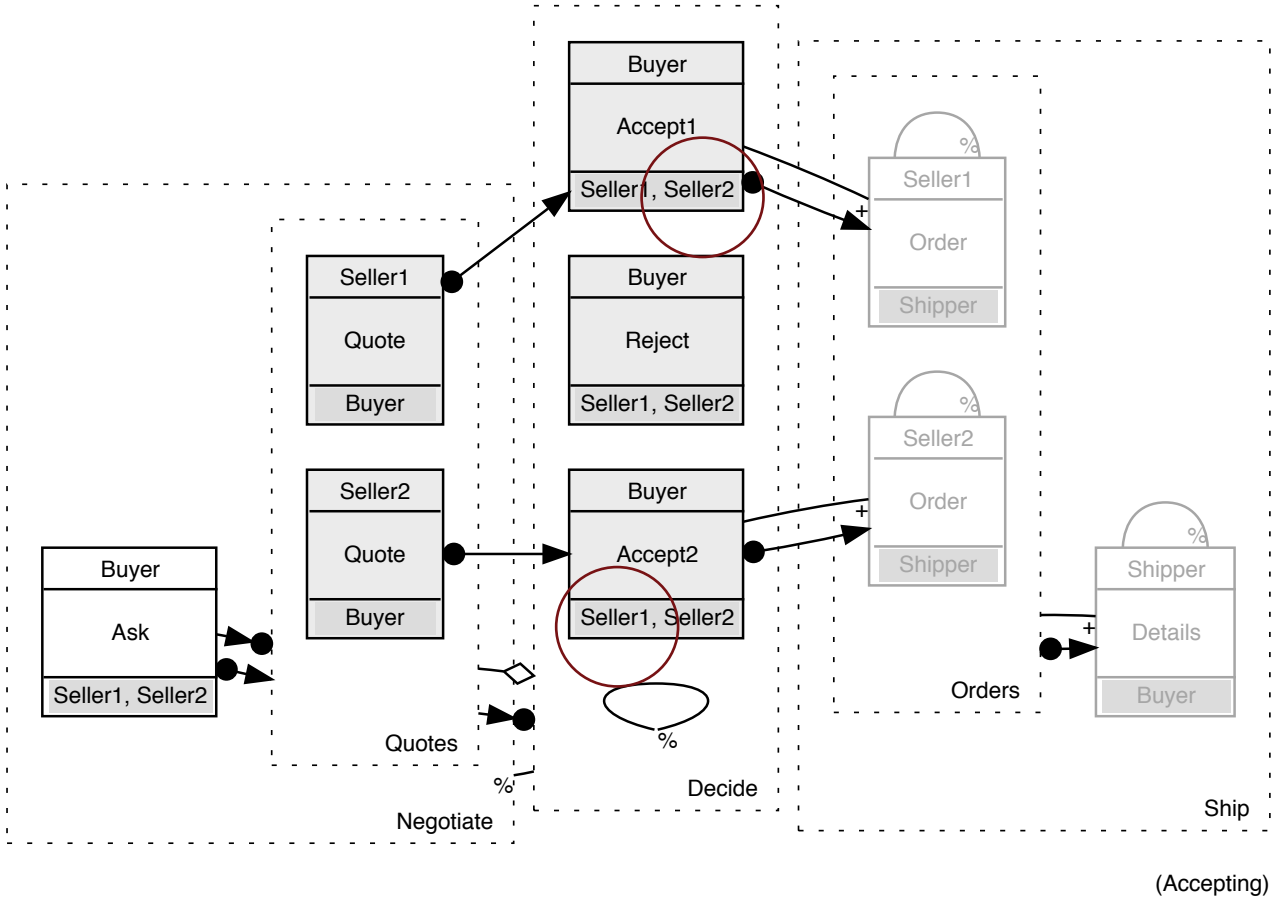


Fig. 4. End-point projectable DCR choreography

Review questions

- What are the basic actions of a choreography model ?

  Interactions

- What is the point of making choregoraphy models?

  Overview of interactions in distributed system and deadlock freedom

- What is an end-point projection?

  The projection of a choreography to individual end point processes/services

- Can we always make an end-point projection ?

  No, we may introduce global dependencies in choreography models!

- What is the point of declarative choreography model as a DCR Graph ?

  More flexible specification closer to requirments, can also describe liveness

Mandatory assignment 2:

Part 1:

In Fig. 1, we show a BPMN choreography inspired from [3]. 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.

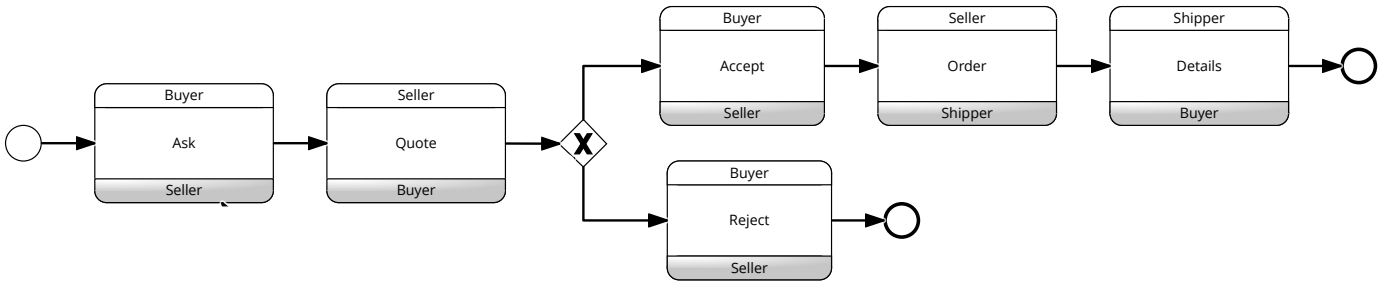
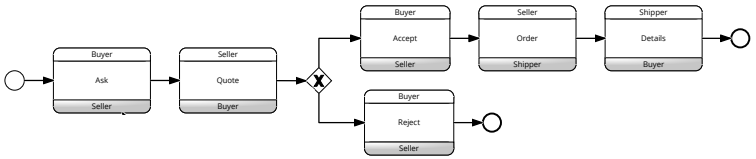


Fig. 1. BPMN Choreography for Buyer-Seller-Shipper example.

Modelling the end-points in CCS



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 (part 1)

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: Implementing the Choreography in Jolie

```
    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)
            }
        }
    }
}
```

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

Implementation of BuyerService

Exercises (Part 2)

2.1 Implement the Seller and the Shipper in Jolie 1.10. You may implemented the relaxed Seller mentioned in Exercises 1.3. You may also allow the seller and shipper to accept multiple concurrent requests.

2.2 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.

2.3 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.