choreology

Coordination and WS-CDL:

Reliable foundation for Business Protocols

- Tony Fletcher
- Peter Furniss
- Bob Haugen

choreology

Proposed changes to WS-CDL:

- Coordinating a single-level choreography:
  - Add “coordination” attribute to choreography.
  - Add <throw/> or <raise/> activity to trigger exception block.
- Coordinating inner choreographies:
  - Allow multiple finalizers, distinguished by attribute “case”.
  - Add <finalize /> activity to identify when to fire which finalizer.
  - Add attribute to <perform /> to label inner choreography instance.
- Composing or “overlaying” coordination:
  - Add to the composition mechanism (perform) the ability to “overlay” choreographies.

choreology

Why to add these features to CDL:

So that CDL can become the language
for expressing
interoperable business protocols
like TWIST.

choreology

Interoperable Business Protocols

- TWIST implementations will need to interoperate.
- Plus, TWIST will need to interoperate with other B2B specs like FIX and SWIFT.
- Are any of them specified consistently enough for implementations to reliably interoperate?
  - No, interop is accomplished by trial-and-error…
- WS-CDL could help define a new generation of reliable business protocol specs.
- But WS-CDL would need business transactions.

choreology

Multiple finalizer cases:

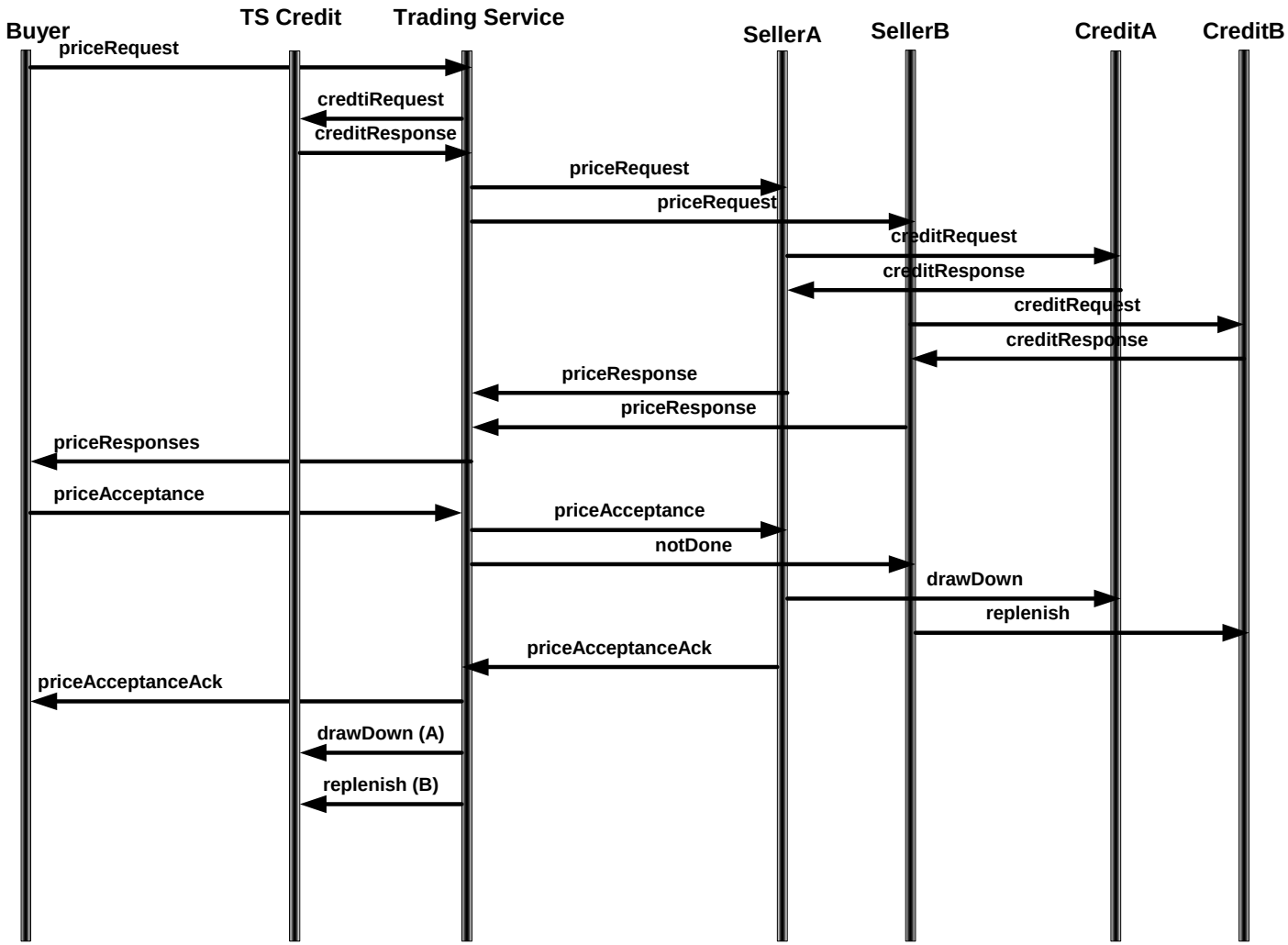
- For example, “confirm” and “cancel”, or “draw down” and “replenish”.
- Because TWIST can’t use compensation
- …and the same will be true for most business protocols.

choreology

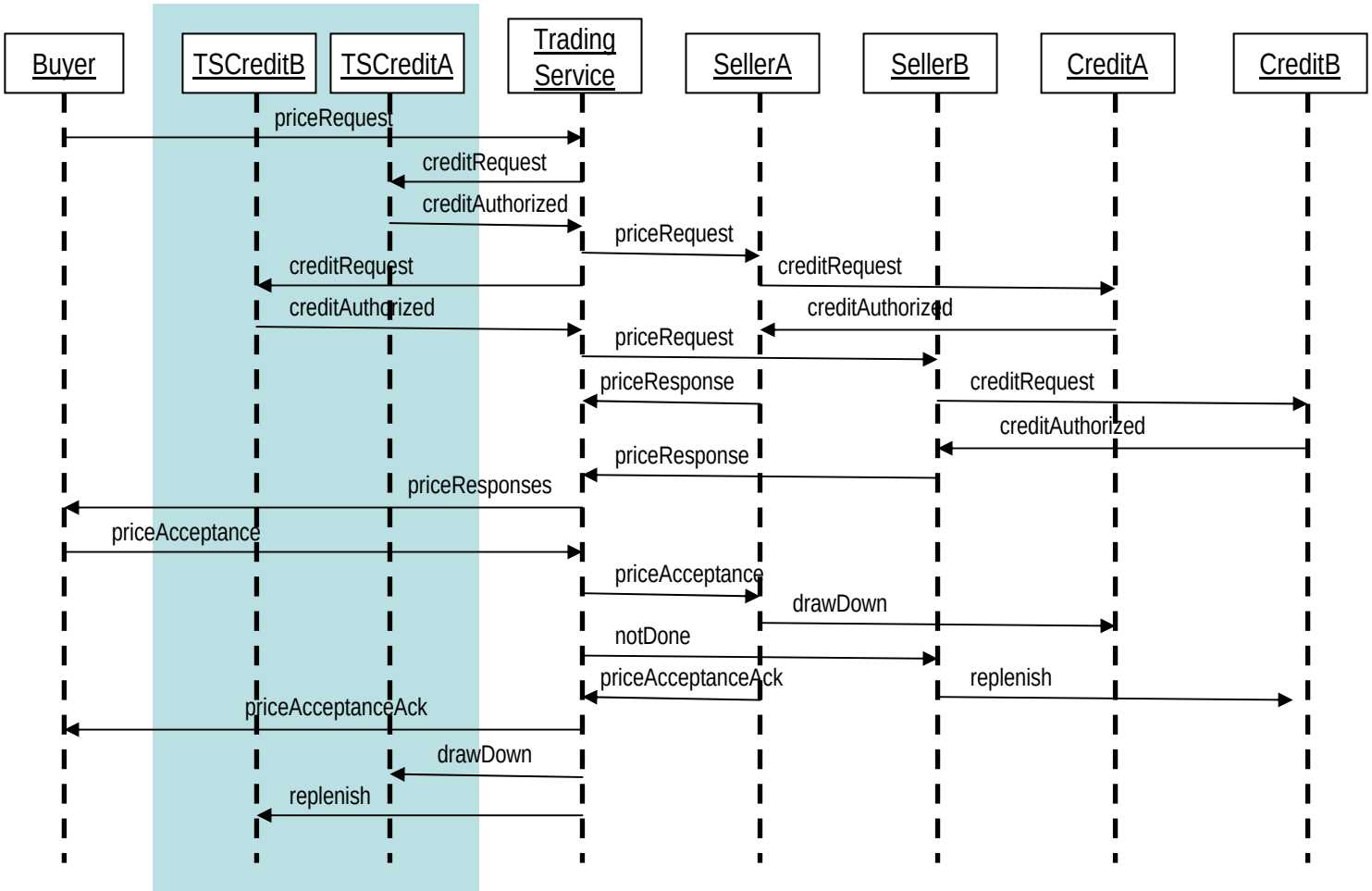
You don’t need to use all of these features together…

…but TWIST could.

Original TWIST scenario 7.2.7

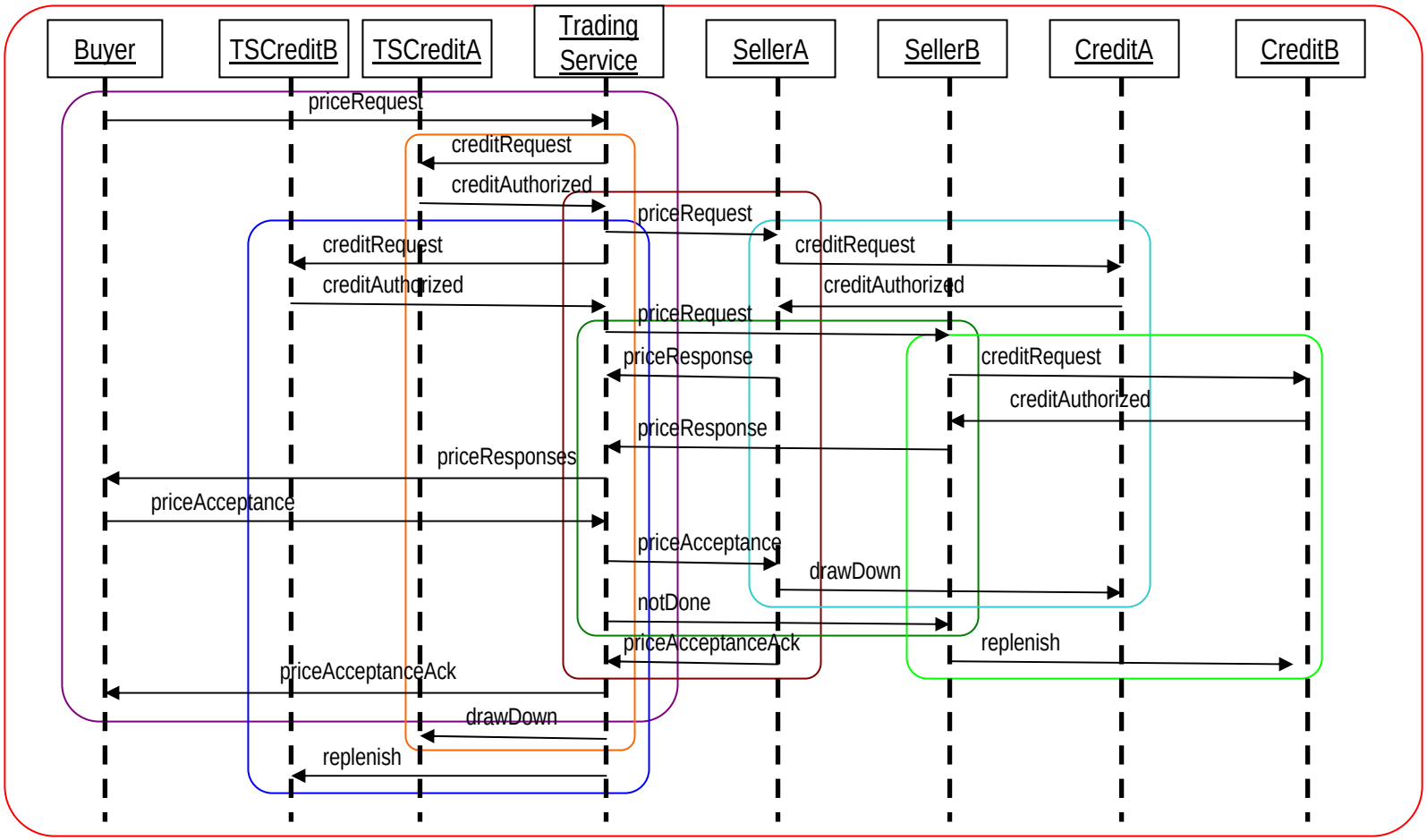


Adding two TSCredit roles, for Sellers A and B



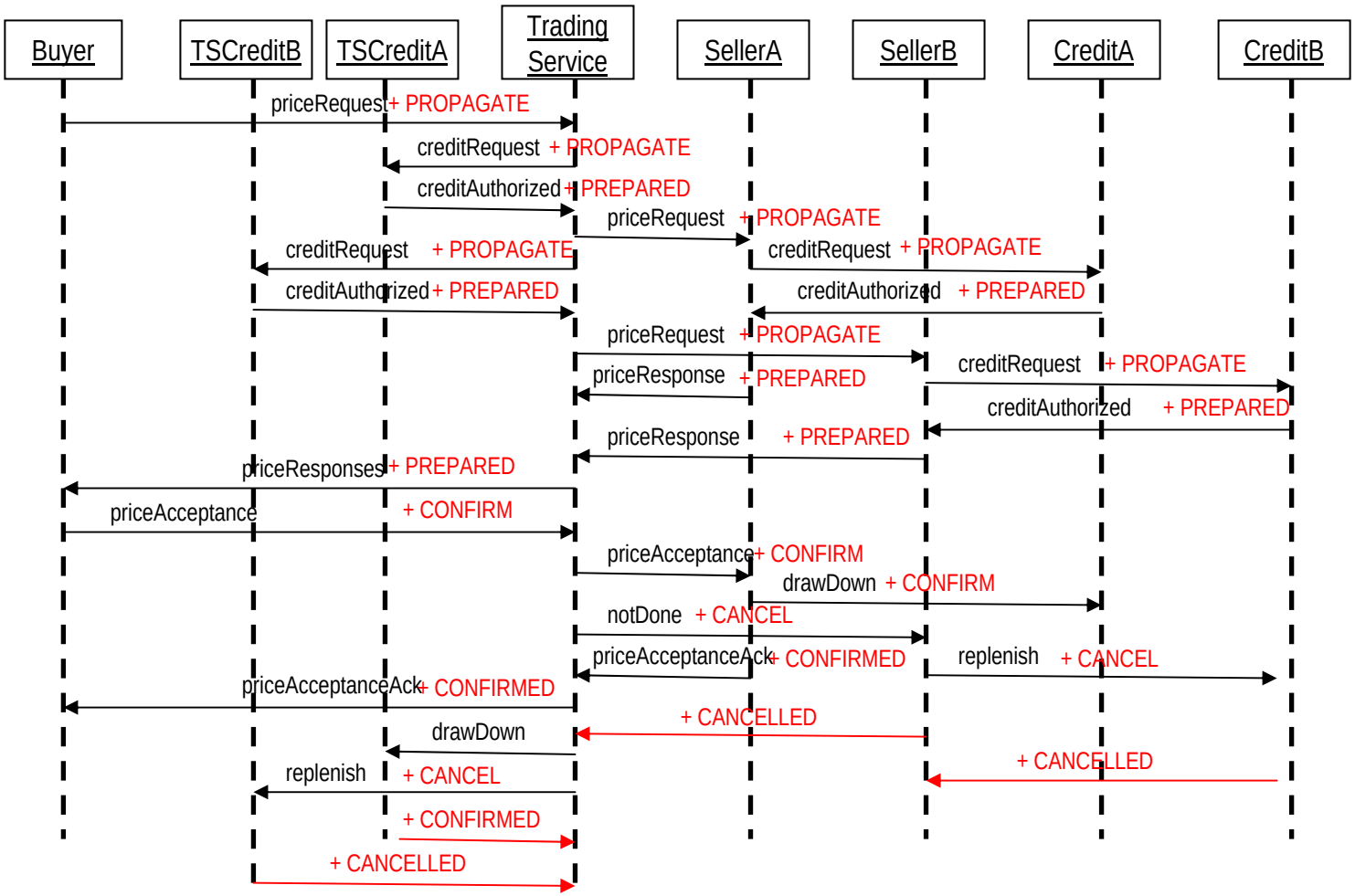
choreology

With transaction boundaries



choreology

With transaction signals

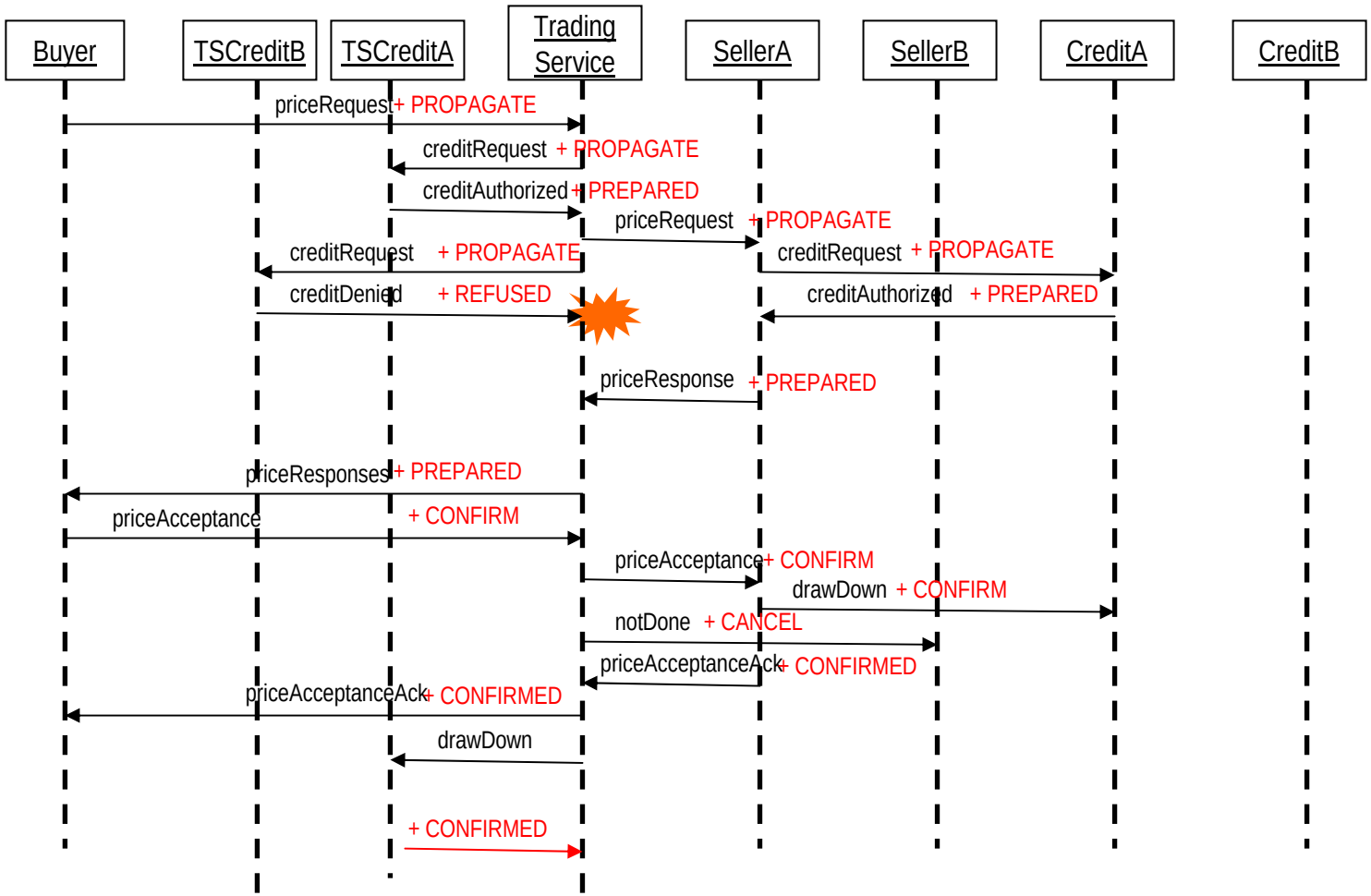


choreology

Each transaction follows the same pattern:

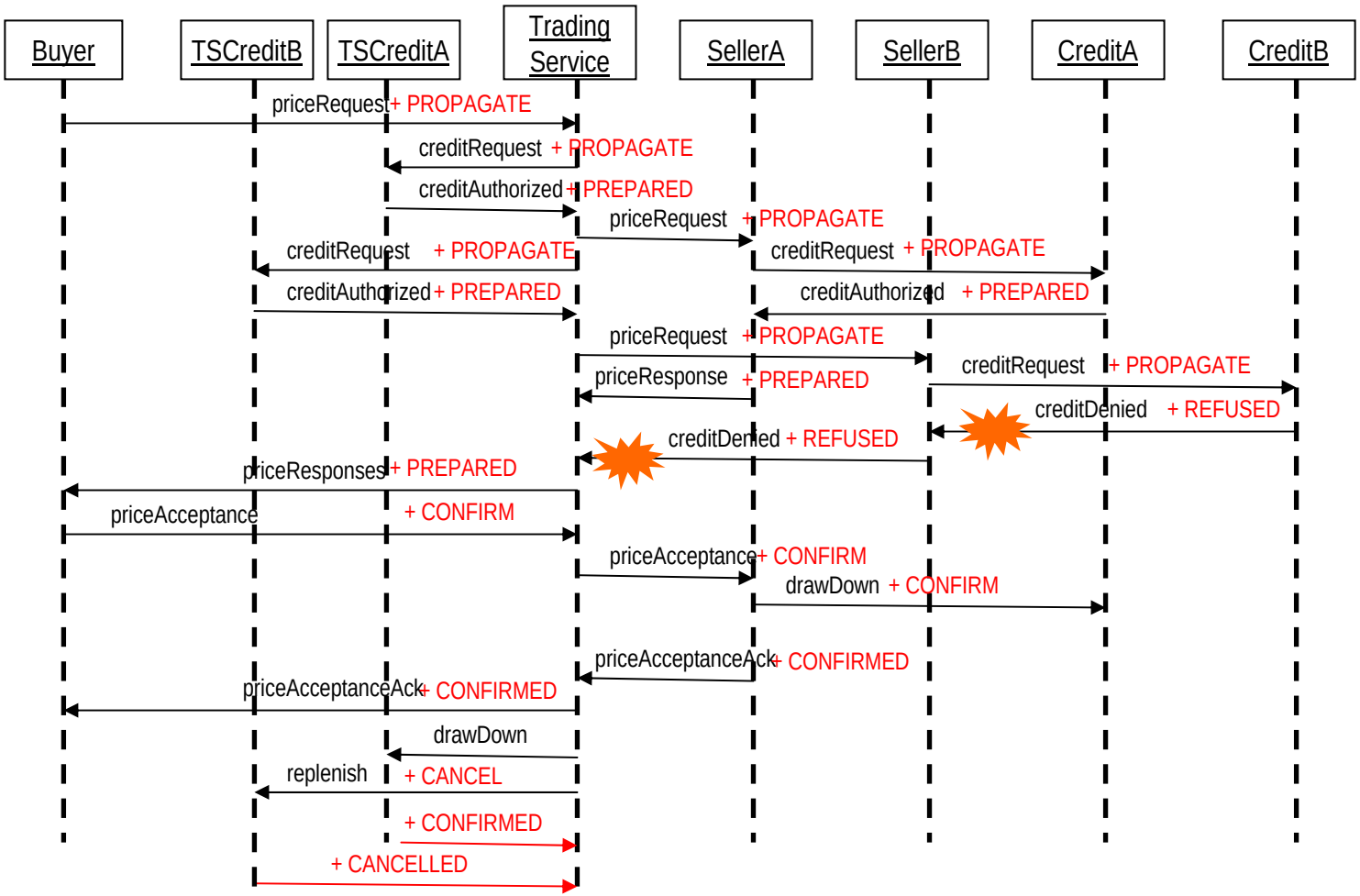
- App Request + PROPAGATE transaction context
  - In TWIST, that's a "Conversation".
- App Provisional Response + PREPARED
- App Conclusion + CONFIRM or CANCEL the provisional response
- (optional App Ack) + CONFIRMED or CANCELLED

Something goes wrong: creditDenied for SellerB



choreology

Something else goes wrong: CreditB denies credit to Buyer



Business Transaction Management Software for Application Coordination

choreology

Revised pattern:

- App Request + PROPAGATE transaction context
- Choice:
  - App refuses request + REFUSED
  - Sequence:
    - App Provisional Response + PREPARED
    - App Conclusion + CONFIRM or CANCEL the provisional response
    - (optional App Ack) + CONFIRMED or CANCELLED

*Note: the above is **not** the CDL for the pattern.*

choreology

How to implement the pattern in CDL:

- Separate the coordination protocol and application messages into different choreographies.
- The same coordination protocol choreography overlays each of the application choreographies.
- *We use a simplified generic protocol that fits TWIST in these examples, but you could substitute any applicable business transaction protocol*.

choreology

Coordination Choreography

```
<choreography name="Coordination" root="false" coordinated="true">
  <sequence>
    <interaction channelVariable="tns:coordination" operation="propagate"
    initiateChoreography="true">
    </interaction>
    <choice>
      <sequence>
          <interaction channelVariable="tns:coordination" operation="prepared“>
          </interaction>
      </sequence>
      <throwException name=“refused" />
    </choice>
  </sequence>
  <exception>
     <interaction channelVariable="tns:coordination" operation="refused“>
     </interaction>
  </exception>
  <finalizer case="confirm">
     <interaction channelVariable="tns:coordination" operation="confirm“>
     </interaction>
  </finalizer>
  <finalizer case="cancel">
     <interaction channelVariable="tns:coordination" operation="cancel“>
     </interaction>
  </finalizer>
</choreography>
```

choreology

Credit Check Choreography

```
<choreography name="CreditCheck" root="false" coordinated="true">
 <sequence>
  <interaction channelVariable="tns:CreditRequestor" operation="creditRequest" initiateChoreography="true">
  </interaction>
  <choice>
   <sequence>
    <interaction channelVariable="tns:CreditResponder" operation="creditAuthorized"
    initiateChoreography="false">
    </interaction>
   </sequence>
   <throwException name="creditDenied" />
  </choice>
 </sequence>
 <exception>
  <interaction channelVariable="tns:CreditResponder" operation="creditDenied" initiateChoreography="false">
  </interaction>
 </exception>
 <finalizer case="confirm">
  <interaction channelVariable="tns:CreditRequestor" operation="drawDown" initiateChoreography="false">
  </interaction>
 </finalizer
 <finalizer case="cancel">
  <interaction channelVariable="tns:CreditRequestor" operation="replenish" initiateChoreography="false">
  </interaction>
 </finalizer>
</choreography>
```

Coordinated Credit Check Choreography

```
<choreography name="CoordinatedCreditCheck" root="false">
  <!-- declare roles, messages and variables     -->
  <parallel>

      <perform choreographyName="Coordination">
            <bind>
               <!-- bind the roles     -->
               <!-- bind the operations     -->
               <!-- bind the variables     -->
            </bind>
      </perform>

      <perform choreographyName="CreditCheck">
            <bind>
               <!-- bind the roles     -->
               <!-- bind the operations     -->
               <!-- bind the variables     -->
            </bind>
      </perform>

  </parallel>
</choreography>
```

choreology

Enclosing Choreography

```
<choreography name="Enclosing" root="true"> <!-- only a snippet shown -->
 <sequence> <!-- other stuff here -->
  <parallel>
   <perform name="checkCreditForA" choreographyName="CoordinatedCreditCheck">
   </perform>
   <perform name="checkCreditForB" choreographyName="CoordinatedCreditCheck">
   </perform>
  </parallel> <!-- other stuff here -->
  <choice>
   <parallel> <!– A was chosen -->
    <finalize name="checkCreditForA" case="confirm" />
    <finalize name="checkCreditForB" case="cancel" />
   </parallel>
   <parallel> <!– B was chosen -->
    <finalize name="checkCreditForA" case="cancel" />
    <finalize name="checkCreditForB" case="confirm" />
   </parallel>
   <parallel> <!– Neither was chosen -->
    <finalize name="checkCreditForA" case="cancel" />
    <finalize name="checkCreditForB" case="cancel" />
   </parallel>
  </choice>
 </sequence>
</choreography>
```

choreology

What does the pattern buy you? Part 1

- The pattern helps to design choreographies that work.
- The coordination choreography includes required protocol signals.
- The application choreographies supply the business messages.
- By means of the protocol signals, the application choreographies can now be managed by business transaction management software.

choreology

What does the pattern buy you? Part 2: exception handling

- Business analysts hate to (often refuse to) handle exceptions.
  - Witness the TWIST scenarios…
- Exceptions in business coordination scenarios multiply like rabbits.
- A good business transaction protocol will handle many exceptions, eg:
  - Recover and realign regardless of end-system or comms failures.
  - Resolve collisions in a defined manner.
  - Handle timeouts.
  - Handle either side deciding to give up (i.e. app triggered exception) at any time.
- Good business transaction management software will handle all protocol exceptions.
- Most of those exceptions can safely be ignored by the business analysts.

choreology

Conclusion

- We think this proposal is more conceptually compatible with CDL than the previous transaction proposal.
  - No surprise, CDL has evolved a lot, and we've thought about it a lot more.
- Not many changes to CDL, and most of them forced moves: they'll need to be done anyway.
- Useful in simple single-level choreographies;
- more useful as the choreographies get more complex.