Session-based Choreography with Exceptions

Marco Carbone

Queen Mary University of London

Places 2008 - 7/6/08 Oslo, Norway
What is this talk about?

- Exceptions and Choreography
- Syntax/Semantics for calculus with exceptions
- Types for exceptions
- Examples and End-Point Projection
Choreography

• Idea from WS-CDL (Choreography Description Language from W3C)

• Communication as a central aspect

• “Dancers dance following a global scenario without a single point of control” (WS-CDL W3C Working Group)

• The programmer (often the architect) describes which two peers must interact without describing the single behaviour of each one of them
A simple system (in $d\pi$)

**Buyer**
call $a(s)$. $s?(x)$. if ($x<100$) then $s!<ok>$ else $s!<no>$

**Broker**
service $a(s)$. call $b(t)$. $t?(x)$. $s!<x+10>$. $s?(y)$. $t!<y>$

**Seller**
service $b(t)$. $t!<quote>$. $t?(x)$
A Choreography for the same system

Buyer $\rightarrow$ Broker : $a(s)$.
Broker $\rightarrow$ Seller : $b(t)$.
Seller $\rightarrow$ Broker $t<\text{quote},x>$. 
Broker $\rightarrow$ Buyer $s<x+10,x>$. 
if ($x<100$) then 
Buyer $\rightarrow$ Broker $s<\text{ok}>$. Broker $\rightarrow$ Seller $<\text{ok}>$ 
else 
Buyer $\rightarrow$ Broker $s<\text{no}>$. Broker $\rightarrow$ Seller $<\text{no}>$
What is an Exception?

• “An Exception is a person or thing that is excluded from a general statement or does not follow a rule” (Mac Dictionary)

• “Exception (handling) is a programming language construct or computer hardware mechanism designed to handle the occurrence of some condition that changes the normal flow of execution.” (Wikipedia)
Exceptions, in general

```java
try { /* Default Code */ }
catch { /* Handler Code */ }
```

• If an exception is thrown by the default code then the handler is executed.

• Exceptions are thrown with a special command `throw`
Choreography with Exceptions

Buyer $\rightarrow$ Seller : $\text{chSeller}(s)$ [s, rec X. Seller $\rightarrow$ Buyer : s <update,quote,x>. if (x<100) then throw else X,]

Seller $\rightarrow$ Buyer : s <conf,cnum,x>. ]

Buyer $\rightarrow$ Seller : s<data,credit,x> ]

} try

} catch
# Syntax

<table>
<thead>
<tr>
<th>1, J ::=</th>
<th>A → B : b(s)[ ^t, 1, J ]</th>
<th>(init)</th>
</tr>
</thead>
<tbody>
<tr>
<td>try (~s) { 1 } catch {J}</td>
<td>(try-catch)</td>
<td></td>
</tr>
<tr>
<td>throw</td>
<td>(throw)</td>
<td></td>
</tr>
<tr>
<td>{ { J } }</td>
<td>(wrap)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A → B : s⟨op, e, y⟩ . 1</th>
<th>(com)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J</td>
</tr>
<tr>
<td>if e@A then 1 else J</td>
<td>(cond)</td>
</tr>
</tbody>
</table>

| 0, 1+J, (v s)l, rec X.l, X | (others) |
Semantics

\[(A \rightarrow B : s\langle op, e, y \rangle . \ i, \ \sigma) \Rightarrow (i, \ \sigma[x@B:=e])\]

(\(\sigma\) is a state as in imperative languages)

\[A \rightarrow B : b(s)[\sim t, \ i, \ J] \Rightarrow (vs) (\text{try} \{ \ i \} \ \text{catch} \{ \ J \})\]

\[i \rightarrow (i', \ S)\]

\[\text{try} (\sim t) \{ \ \text{throw} \mid i \} \ \text{catch} \{ \ J \} \Rightarrow \{ \{ \ J \} \}\]
End-point Projection

```plaintext

call chSeller(s)[
    μX. s?(y). if ok(y) then throw else X, try
    s!<card>. s?(z) ] catch
]

service chSeller(s)[
    μX. s<quote>. X, try
    s?(x).s!<time> ] catch
]
```

Interactional Exceptions

• When an exception is raised both parties in a conversation must change the flow of their execution

• The end-point projection above requires a propagation mechanism at end-point
Session Types?

• “Types are almost standard”

• We need to add a new type i.e. an abstraction for

  \[
  \textbf{try} \ (\sim s) \ \{ \ I \ \} \ \textbf{catch} \ \{ J \}
  \]

• A try-catch type $\alpha \ {\{ \ \beta \ \}}$ where $\alpha$ is the type for $I$ and $\beta$ the type for $J$
Choreography with Exceptions

Buyer → Broker : chBroker(s) [ s,

   Buyer → Broker : s⟨identify, id, x⟩.
   if bad(x) then throw
else Broker → Seller : chSeller(t)[( s, t),
   rec X. Seller → Broker : t⟨update, quote, y⟩.
   Broker → Buyer : s⟨update, y + 10%, y⟩.
   if (y < 100) then throw else X,

Seller → Broker : t⟨conf, cnum, x⟩.
Broker → Buyer : s⟨conf, x, x⟩.
Buyer → Broker : s⟨data, credit, x⟩.
Broker → Seller : t⟨data, x, x⟩],

Seller → Buyer : s⟨reject, reason, x⟩. l ]
End-point Projection

\textbf{chBroker}(s)[ s,  
  s!<id>. \textbf{rec} X . s?(y). \textbf{if} ok(y) \textbf{throw} \textit{else} X, 
  \textbf{conf}: s?(x). s!<credit> + \textbf{reject}: s?(x). P ]

\*\textbf{chBroker}(s)[ s,  
  s?(x). \textbf{if} bad(x) \textbf{then} \textbf{throw}  
  \textit{else} \textbf{chSeller}(t)[ (t,s), [\ldots\text{forwarding}\ldots]],  
  \textbf{select} s(\text{conf}) [\ldots\text{forwarding}\ldots],  
  \textbf{select} s(\text{reject}) : P ]
Conclusions, To Do’s

• Exceptions for Choreography are *naturally interactional*.

• The propagation mechanism is necessary!

• Interactional Exceptions for end-points with a built-in propagation mechanism have been studied in [5] (to appear in CONCUR08)

• Next step is to formally study the end-point projection:
  – Can we relax/Shall we strengthen the conditions in [4] (EPP)
  – Will EPP be sound and complete wrt to semantics and typing?