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 dπ)

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 → **Broker** : a(s).
Broker → **Seller** : b(t).
Seller → **Broker** t<quote,x>.
Broker → **Buyer** s<x+10,x>.
if (x<100) **then**
 Buyer → **Broker** s<ok>. **Broker** → **Seller**<ok>
else
 Buyer → **Broker** s<no>. **Broker** → **Seller** <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

```
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 → Seller : chSeller(s) [s,

rec X. **Seller → Buyer** : s <*update*,quote,x>.
if (x<100) then throw else X,

} try

Seller → Buyer : s <*conf*,cnum,x>.
Buyer → Seller : s<*data*,credit,x>]

} catch

Syntax

I , J ::=	**A → B : b(s)[˜t, I , J]**	(init)
	try (˜s) { I } catch {J}	(try-catch)
	throw	(throw)
	{{ J }}	(wrap)
	A → B : s⟨op, e, y⟩ . I	(com)
	I | J	(par)
	if e@A then I else J	(cond)
	0, I+J, (v s)I, rec X.I, X	(others)

Semantics

(A → B : s⟨op, e, y⟩ . I, σ) ➔ (I, σ[x@B:=e])

(σ is a state as in imperative languages)

A → B : b(s)[˜t, I, J] ➔ (vs) (try { I } catch {J})

$$\frac{\mathbf{I \rightarrow (I', S)}}{\mathbf{try\ (\sim t)\ \{\ throw\ |\ I\ \}\ catch\ \{\ J\ \} \rightarrow\ \{\{\ J\ \}\}}}$$

End-point Projection

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

  try (~s) { I } **catch** {J}

- A try-catch type α {{ β }} where α is the type for I and β 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⟩ . I]

End-point Projection

chBroker(s)[s,
  s!<id>. rec X . s?(y). **if** ok(y) **throw** else X,

  conf: s?(x). s!<credit> + reject: s?(x). P]

∗**chBroker**(s)[s,
  s?(x). **if** bad(x) **then throw**
  else chSeller(t)[(t,s), […forwarding…],
        select s(conf) […forwarding…],

  select s(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?