Global Escape in Multiparty Sessions

Sara Capecchi

joint work with **Elena Giachino** & **Nobuko Yoshida**

Workshop on Behavioural Types
21 April 2011

Global escape

Global escape

- unexpected condition, computational error

Global escape

- unexpected condition, computational error
- controlled structured interruption requested by some participant

Global escape

- unexpected condition, computational error
- controlled structured interruption requested by some participant

Interactional exceptions (Structured Interactional Exceptions for Session Types. Carbone, Honda, Yoshida. CONCUR'08)

not only local but also coordinated actions among communicating peers: exception affects a collection of parallel processes and an escape needs to move into another dialogue in a concerted manner

Goals & Issues

Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;

Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
- preserve multiparty session properties:

  Subject Reduction *Communication Safety* *Session Fidelity*
  Progress

Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
- preserve multiparty session properties:

  Subject Reduction *Communication Safety* *Session Fidelity*
  Progress
- how to model

Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
- preserve multiparty session properties:

  Subject Reduction *Communication Safety* *Session Fidelity*
  Progress

- how to model
  - concurrent exceptions

Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
- preserve multiparty session properties:

  Subject Reduction *Communication Safety* *Session Fidelity*
  Progress

- how to model
  - concurrent exceptions
  - asyncronous notification to multiple partners

Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
- preserve multiparty session properties:

  Subject Reduction *Communication Safety* *Session Fidelity*
  Progress
- how to model
  - concurrent exceptions
  - asyncronous notification to multiple partners
  - nested exceptions

Coordinated Actions Model

From *Coordinated Exception handling- Romanovsky et al.*

Fault tolerance needs error isolation to define exactly which part of the system to recover, and to prevent errors from unlimited propagation. One way to control complexity is to *restrict interaction and communication*: exception contexts are defined as regions in which the same exceptions are treated in the same way

Coordinated Actions Model

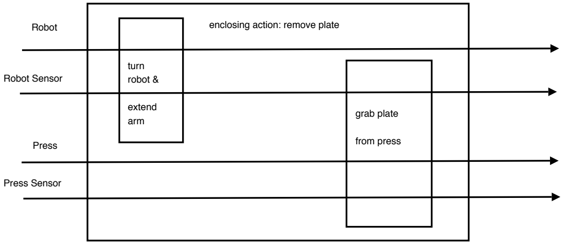
From *Coordinated Exception handling- Romanovsky et al.*

Fault tolerance needs error isolation to define exactly which part of the system to recover, and to prevent errors from unlimited propagation. One way to control complexity is to *restrict interaction and communication*: exception contexts are defined as regions in which the same exceptions are treated in the same way

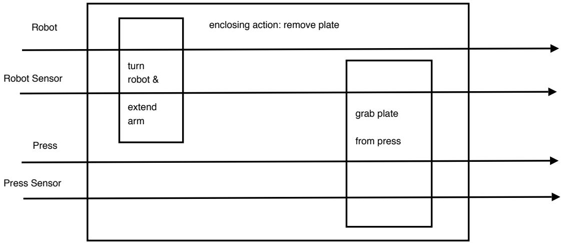
Atomic actions

The activity of a group of components constituites an atomic action if there are no interactions between that group and the rest of the systems for the duration of the activity

Coordinated Actions

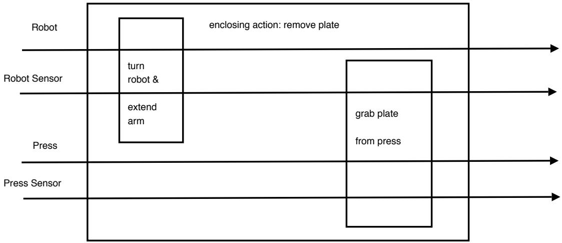


Coordinated Actions



$$\left[(s_1, s_2), \left[s_1, \gamma_{TR}, \gamma_{HTR}\right]; \left[s_1, \gamma_{GP}, \gamma_{HGP}\right], \gamma_{HRP}\right]$$

Coordinated Actions



$$[(s_1, s_2), [s_1, \gamma_{TR}, \gamma_{HTR}]; [s_1, \gamma_{GP}, \gamma_{HGP}], \gamma_{HRP}]$$

$\texttt{Robot} = \mathrm{try}(s_1, s_2)\{\mathrm{try}(s_1)\{P^R\}\ \mathrm{catch}\ \{Q^R\}\}\ \mathrm{catch}\ \{Q'^R\}$
$\texttt{RobotSensor} = \mathrm{try}(s_1, s_2)\{\mathrm{try}(s_1)\{P^{RS}\}\ \mathrm{catch}\ \{Q^{RS}\}; \mathrm{try}(s_1)\{P'_{RS}\}\ \mathrm{catch}\ \{Q'^{RS}\}\}\ \mathrm{catch}\ \{Q''^{RS}\}$
$\texttt{Press} = \mathrm{try}(s_1, s_2)\{\mathrm{try}(s_1)\{P^P\}\ \mathrm{catch}\ \{Q^P\}\}\ \mathrm{catch}\ \{Q'^P\}$
$\texttt{PressSensor} = \mathrm{try}(s_1, s_2)\{\mathrm{try}(s_1)\{P^S\}\ \mathrm{catch}\ \{Q^{PS}\}\}\ \mathrm{catch}\ \{Q'^{PS}\}$

Syntax and Semantics

P, Q	::=	$\overline{a}[2..n](\tilde{s}).P$	Multicast Request
	|	$a[\mathrm{p}](\tilde{s}).P$	Accept
	|	$r!\langle\tilde{e}\rangle$	Output
	|	$r?(\tilde{x}).P$	Input
	|	$r \triangleleft l.P$	Select
	|	$r \triangleright \{l_i : P_i\}_{i \in I}$	Branch
	|	$\mathrm{try}(\tilde{r})\{P\}\ \mathrm{catch}\ \{P\}$	Try-Catch
	|	$\mathrm{throw}(\tilde{r})$	Throw

|	$\mathrm{if}\ e\ \mathrm{then}\ P\ \mathrm{else}\ \ P$	Conditional
|	$P \mid P$	Parallel
|	$P; P$	Sequencing
|	$\mathbf{0}$	Inaction
|	$(\nu n)P$	Hiding
|	$\mathrm{def}\ D\ \mathrm{in}\ P$	Recursion
|	$X\langle\tilde{e}\tilde{s}\rangle$	Process call
|	$s : L$	Named queue

Syntax and Semantics

P,Q	$::=$	$\overline{a}[2..n](\tilde{s}).P$	Multicast Request
	$\mid$	$a[\mathrm{p}](\tilde{s}).P$	Accept
	$\mid$	$r!\langle\tilde{e}\rangle$	Output
	$\mid$	$r?(\tilde{x}).P$	Input
	$\mid$	$r \lhd l.P$	Select
	$\mid$	$r \rhd \{l_i : P_i\}_{i\in I}$	Branch
	$\mid$	$\mathrm{try}(\tilde{r})\{P\}\ \mathrm{catch}\ \{P\}$	Try-Catch
	$\mid$	$\mathrm{throw}(\tilde{r})$	Throw
	$\mid$	$\mathrm{if}\ e\ \mathrm{then}\ P\ \mathrm{else}\ \ P$	Conditional
	$\mid$	$P \mid P$	Parallel
	$\mid$	$P;\ P$	Sequencing
	$\mid$	$\mathbf{0}$	Inaction
	$\mid$	$(\nu n)P$	Hiding
	$\mid$	$\mathrm{def}\ D\ \mathrm{in}\ P$	Recursion
	$\mid$	$X\langle\tilde{e}\tilde{s}\rangle$	Process call
	$\mid$	$s : L$	Named queue

[Thr]

$$\Sigma \vdash \mathrm{try}(\tilde{r})\{\mathcal{C}[\mathrm{throw}(\tilde{r})] \mid P\}\ \mathrm{catch}\ \{Q\}$$
$$\longrightarrow \Sigma \uplus \mathrm{throw}(\tilde{r}) \vdash \mathrm{try}(\tilde{r})\{\mathcal{C} \mid P\}\ \mathrm{catch}\ \{Q\}$$

[RThr]

$$\Sigma, \mathrm{throw}(\tilde{r}) \vdash \mathrm{try}(\tilde{r})\{P\}\ \mathrm{catch}\ \{Q\} \longrightarrow \Sigma, \mathrm{throw}(\tilde{r}) \vdash Q\{s^{\varphi+1}/s^{\varphi}\}_{s^{\varphi}\in\tilde{r}}$$

$(\mathrm{throw}(\tilde{r}')\in\Sigma$ *implies* $\mathrm{try}(\tilde{r}')\ldots\notin P,\ \tilde{r}'\subseteq\tilde{r})$

[ZThr]

$$\Sigma \vdash (\nu\tilde{s})(\textstyle\prod_i \mathcal{E}_i[\mathrm{try}(\tilde{r})\{\mathbf{0}\}\ \mathrm{catch}\ \{Q_i\}])_{i\in 1..n} \longrightarrow \Sigma \vdash (\nu\tilde{s})(\textstyle\prod_i \mathcal{E}_i)_{i\in 1..n}$$

$(\mathrm{throw}(\tilde{r})\notin\Sigma)$

Typing

$$
\begin{array}{llcl}
\text{Partial} & \gamma & ::= & \mathrm{p}_1 \to \mathrm{p}_2 : k\langle \tilde{S} \rangle \mid \mathrm{p}_1 \to \mathrm{p}_2 : k\{l_i : \gamma_i\}_{i \in I} \mid \\
 & & & [\tilde{k}, \gamma, \gamma] \mid \gamma; \gamma \mid \gamma \parallel \gamma \mid \mu \mathbf{t}.\gamma \mid \mathbf{t} \\
\text{Global} & G & ::= & \gamma; \text{end} \mid \text{end} \\
\text{Sorts} & S & ::= & \text{bool} \mid \ldots \mid \langle G \rangle
\end{array}
$$

Goals:

Typing

$$\begin{array}{llll}
\text{Partial} & \gamma & ::= & \mathrm{p}_1 \to \mathrm{p}_2 : k\langle \tilde{S}\rangle \mid \mathrm{p}_1 \to \mathrm{p}_2 : k\{l_i : \gamma_i\}_{i\in I} \mid \\
 & & & [\tilde{k}, \gamma, \gamma] \mid \gamma; \gamma \mid \gamma \parallel \gamma \mid \mu\mathbf{t}.\gamma \mid \mathbf{t} \\
\text{Global} & G & ::= & \gamma; \text{end} \mid \text{end} \\
\text{Sorts} & S & ::= & \text{bool} \mid \ldots \mid \langle G\rangle
\end{array}$$

Goals:

- to check that the enclosed try-catch block is listening on a smaller set of channels: independence of the components w.r.t. exceptions

Typing

$$\begin{array}{llll} \text{Partial} & \gamma & ::= & \mathrm{p}_1 \to \mathrm{p}_2 : k\langle \tilde{S}\rangle \mid \mathrm{p}_1 \to \mathrm{p}_2 : k\{l_i : \gamma_i\}_{i\in I} \mid \\ &&& [\tilde{k},\gamma,\gamma] \mid \gamma;\gamma \mid \gamma \parallel \gamma \mid \mu\mathbf{t}.\gamma \mid \mathbf{t} \\ \text{Global} & G & ::= & \gamma; \text{end} \mid \text{end} \\ \text{Sorts} & S & ::= & \text{bool} \mid \ldots \mid \langle G\rangle \end{array}$$

Goals:

- to check that the enclosed try-catch block is listening on a smaller set of channels: independence of the components w.r.t. exceptions
- to check that no session request or accept occurs inside a try-catch block

Conclusions

Our extension is:

Conclusions

Our extension is:

- consistent: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix

Conclusions

Our extension is:

- consistent: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix
- safe: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types

Conclusions

Our extension is:

- consistent: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix
- safe: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types

We ensure these properties using:

Conclusions

Our extension is:

- consistent: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix
- safe: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types

We ensure these properties using:

- an asynchronous linguistic construct for exceptions signalling

Conclusions

Our extension is:

- consistent: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix
- safe: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types

We ensure these properties using:

- an asynchronous linguistic construct for exceptions signalling
- multi-level queues