Self-Adaptive Monitors for Multiparty Sessions

joint work with Mario Coppo and Betti Venneri

London, 10 January 2014
1. Introduction

2. A Self-Adaptive System

3. Conclusion
What is adaptation?

we define adaptation as the run-time modification of the control data and a component is self-adaptive if it is able to modify its own control data at run-time.
What is adaptation?

What is adaptation?


“we define adaptation as the run-time modification of the control data ...and a component is self-adaptive if it is able to modify its own control data at run-time”
What is adaptation?


“we define adaptation as the run-time modification of the control data ...and a component is self-adaptive if it is able to modify its own control data at run-time”

we need to distinguish between standard data and control data: a change in the system behaviour is part of the application logic if it is based on standard data, it is an adaptation if it is based on control data.
Scenarios

- A community has many distributed entities which interact with each other according to a given operational plan,
Scenarios

- a community has many distributed entities which interact with each other according to a given operational plan,
- the complex dynamic environment can present unforeseen events, which require the community to modify its plan dynamically,
Scenarios

- A community has many distributed entities which interact with each other according to a given operational plan,
- The complex dynamic environment can present unforeseen events, which require the community to modify its plan dynamically,
- Those critical events are observed in any separate component of the system, which can be checked by the session participants, so that the whole system can react promptly by updating itself,
Scenarios

- a community has many distributed entities which interact with each other according to a given operational plan,
- the complex dynamic environment can present unforeseen events, which require the community to modify its plan dynamically,
- those critical events are observed in any separate component of the system, which can be checked by the session participants, so that the whole system can react promptly by updating itself,
- the dynamic changes need to be rather flexible: in each adaptation phase, new participants can be introduced or some of the old participants are not longer involved (temporarily or permanently),
Scenarios

- a community has many distributed entities which interact with each other according to a given operational plan,
- the complex dynamic environment can present unforeseen events, which require the community to modify its plan dynamically,
- those critical events are observed in any separate component of the system, which can be checked by the session participants, so that the whole system can react promptly by updating itself,
- the dynamic changes need to be rather flexible: in each adaptation phase, new participants can be introduced or some of the old participants are not longer involved (temporarily or permanently),
- these dynamic changes need to be safe: community interactions must proceed correctly to pursue the common task.
Example
Example
Example
Global Type

\[ G_1 = iS \rightarrow iF: (\text{Item}, \text{Amount}) \]

\[ bS \rightarrow bF: (\text{Item}, \text{Amount}) \]

\[ \text{Ada} \rightarrow \{ iS, iF, bS, bF \} : \text{check} \]
Global Type

\[ G_1 = \begin{align*}
    iS & \rightarrow iF : (\text{Item}, \text{Amount}). \\
    bS & \rightarrow bF : (\text{Item}, \text{Amount}). \\
    \text{Ada} & \rightarrow \{iS, iF, bS, bF\} : \text{check}
\end{align*} \]
Example
Example
Global Type

Outline
- Introduction
- A Self-Adaptive System
- Conclusion

Global Type

G\rightarrow^2 Ada \rightarrow Bob:Contract

iS \rightarrow iF:(Item, Amount)

bS \rightarrow Ada:(Item, Amount)

Ada \rightarrow \{iS, iF, bS, Bob\}: check
Global Type

\[ G_2 = \]

\[
\begin{align*}
\text{Ada} & \rightarrow \text{Bob} : \text{Contract}. \\
\text{iS} & \rightarrow \text{iF} : (\text{Item, Amount}). \\
\text{bS} & \rightarrow \text{Ada} : (\text{Item, Amount}). \\
\text{Ada} & \rightarrow \{\text{iS, iF, bS, Bob}\} : \text{check}
\end{align*}
\]
Example

\[ iS \rightarrow iF : (\text{Item, Amount}). \]
\[ bS \rightarrow bF : (\text{Item, Amount}). \]
\[ \text{Ada} \rightarrow \{ iS, iF, bS, bF \} : \text{check} \]
Monitors

\[ iS \rightarrow iF : (\text{Item}, \text{Amount}). \]
\[ \quad G_1 = bS \rightarrow bF : (\text{Item}, \text{Amount}). \]
\[ \quad \text{Ada} \rightarrow \{iS, iF, bS, bF\} : \text{check} \]
Monitors

\[ iS \rightarrow iF : (\text{Item, Amount}). \]

\[ G_1 = bS \rightarrow bF : (\text{Item, Amount}). \]

\[ \text{Ada} \rightarrow \{iS, iF, bS, bF\} : \text{check} \]

\[ iF!(\text{Item, Amount}).\text{Ada? check} \]

\[ bF!(\text{Item, Amount}).\text{Ada? check} \]
Monitors

\[ iS \rightarrow iF : (\text{Item}, \text{Amount}). \]

\[ G_1 = bS \rightarrow bF : (\text{Item}, \text{Amount}). \]

\[ \text{Ada} \rightarrow \{iS, iF, bS, bF\} : check \]

- iF!(\text{Item}, \text{Amount}).Ada? check
- bF!(\text{Item}, \text{Amount}).Ada? check
- iS?(\text{Item}, \text{Amount}).Ada? check
- bS?(\text{Item}, \text{Amount}).Ada? check
Monitors

\[ iS \rightarrow iF: (\text{Item, Amount}). \]
\[ G_1 = bS \rightarrow bF: (\text{Item, Amount}). \]
\[ Ada \rightarrow \{ iS, iF, bS, bF \}: \text{check} \]

\[ iF!(\text{Item, Amount}). Ada? \text{check} \]
\[ bF!(\text{Item, Amount}). Ada? \text{check} \]

\[ iS?(\text{Item, Amount}). Ada? \text{check} \]
\[ bS?(\text{Item, Amount}). Ada? \text{check} \]

\[ \{ iS, iF, bS, bF \}! \text{check} \]
Monitors

\[ G_2 = \]

\[
\text{Ada} \rightarrow \text{Bob} : \text{Contract.}
\]

\[
iS \rightarrow iF : (\text{Item, Amount}).
\]

\[
bS \rightarrow \text{Ada} : (\text{Item, Amount}).
\]

\[
\text{Ada} \rightarrow \{iS, iF, bS, \text{Bob}\} : \text{check}
\]
Monitors

\[ G_2 = \]

\begin{align*}
\text{Ada} & \rightarrow \text{Bob} : \text{Contract.} \\
\text{iS} & \rightarrow \text{iF} : (\text{Item}, \text{Amount}) \\
\text{bS} & \rightarrow \text{Ada} : (\text{Item}, \text{Amount}) \\
\text{Ada} & \rightarrow \{ \text{iS, iF, bS, Bob} \} : \text{check}
\end{align*}

\begin{align*}
\text{iF}(\text{Item, Amount}).\text{Ada} \rightarrow \text{check} \\
\text{Ada}(\text{Item, Amount}).\text{Ada} \rightarrow \text{check}
\end{align*}
Monitors

\[ G_2 = \]

Ada → Bob : Contract.

iS → iF : (Item, Amount).

bS → Ada : (Item, Amount).

Ada → \{ iS, iF, bS, Bob \} : check

\( iF!(Item, Amount).Ada? check \)

\( Ada!(Item, Amount).Ada? check \)

\( iS?(Item, Amount).Ada? check \)
Monitors

\[ G_2 = \]

- Ada → Bob : Contract.
- \( \text{iS} \rightarrow \text{iF} : (\text{Item}, \text{Amount}) \).
- \( \text{bS} \rightarrow \text{Ada} : (\text{Item}, \text{Amount}) \).
- Ada → \{ \text{iS, iF, bS, Bob} \} : check

\( \text{iF}!(\text{Item, Amount}).\text{Ada} ? \text{check} \)

\( \text{Ada}!(\text{Item, Amount}).\text{Ada} ? \text{check} \)

\( \text{iS}?(\text{Item, Amount}).\text{Ada} ? \text{check} \)

\( \text{Bob}!\text{Contract}.\text{bS}?(\text{Item, Amount}).\{ \text{iS, iF, bS, Bob} \}! \text{check} \)
Monitors

\[ G_2 = \]

Ada → Bob : Contract.

\[ iS \rightarrow iF : (\text{Item}, \text{Amount}). \]

bS → Ada : (Item, Amount).

Ada → \{iS, iF, bS, Bob\} : check

iF!(Item, Amount).Ada? check

Ada!(Item, Amount).Ada? check

iS?(Item, Amount).Ada? check

Bob!Contract. bS?(Item, Amount).\{iS, iF, bS, Bob\}! check

μX. y!(item, amount). y?check.X
Processes

\[ \mu X. y!(\text{item, amount}). y? \text{check}. X \]

\[ \mu X. y?(\text{item, amount}). \text{if} \ldots \text{then} y? \text{check}. X \]
\[ \text{else write KO}. y? \text{check} \]
Processes

$$\mu X. y!(\text{item, amount}).y? \text{check}.X$$

$$\mu X. y?(\text{item, amount}).\text{if } \ldots \text{ then } y? \text{check}.X$$

else write KO

$$y? \text{check}$$

$$y? \text{contract}.y?(\text{item, amount}).y! \text{check}(F).X$$

$$+ (\mu X. y! \text{check}(F).X)$$
Processes

\[ \mu X. y!(item, amount).y?check.X \]
\[ \mu X. y?(item, amount).if \ldots then y?check.X \else write KO.y?check \]
\[ (\mu X. y!check(F).X) + (\mu X. y!contract.y?(item, amount).y!check(F).X) \]
\[ \mu X. y?contract.if \ldots then write OK.y?check \else y?check.X \]
Processes

\[ \mu X. y!(\text{item, amount}).y?\text{check}.X \]

\[ \mu X. y?(\text{item, amount}).\text{if } \ldots \text{ then } y?\text{check}.X \]
\[ \text{else write } KO.y?\text{check} \]

\[ (\mu X. y!\text{check}(F).X) \]
\[ + (\mu X. y!\text{contract}.y?\text{(item, amount)}.y!\text{check}(F).X) \]

\[ \mu X. y?\text{contract} \text{.if } \ldots \text{ then write } OK.y?\text{check} \]
\[ \text{else } y?\text{check}.X \]

\[ F(\text{OK, OK}) = G_1 \]
\[ F(\text{OK, KO}) = G_2 \]
\[ F(\text{KO, OK}) = G_3 \]
\[ F(\text{KO, KO}) = G_4 \]
System

\[ iF!(\text{Item, Amount}) . \text{Ada?check} \quad [\mu X . y!(\text{item, amount}).y?\text{check}.X] | \]
System

\[ iF!(\text{Item, Amount}).\text{Ada? check} \]  
\[ bF!(\text{Item, Amount}).\text{Ada? check} \]

\[ [\mu X. y!(\text{item, amount}).y? check.X] \]

\[ [\mu X. y!(\text{item, amount}).y? check.X] \]
System

- `iS?(Item, Amount).Ada?check` \( [\mu X. y?(item, amount).if \ldots \text{then } y?check.X \text{ else write } KO.y?check] \)
System

- ![iF!(Item, Amount).Ada?check](image)
  \[\mu X. y!(item, amount). y? check.X\] | 

- ![bF!(Item, Amount).Ada?check](image)
  \[\mu X. y!(item, amount). y? check.X\] | 

- ![iS?(Item, Amount).Ada?check](image)
  \[\mu X. y?(item, amount). if \ldots 
  \quad \text{then } y? check.X 
  \quad \text{else write } KO.y? check\] | 

- ![bS?(Item, Amount).Ada?check](image)
  \[\mu X. y?(item, amount). if \ldots 
  \quad \text{then } y? check.X 
  \quad \text{else write } KO.y? check\] |
System

- \( \text{iF!(Item, Amount).Ada?check} \)  
  \( [\mu X. y!(item, amount).y?\text{check}.X] \)  
- \( \text{bF!(Item, Amount).Ada?check} \)  
  \( [\mu X. y!(item, amount).y?\text{check}.X] \)  
- \( \text{iS?(Item, Amount).Ada?check} \)  
  \( [\mu X. y?(item, amount).\text{if} \ldots \text{then } y?\text{check}.X \text{ else write KO.y?\text{check}}] \)  
- \( \text{bS?(Item, Amount).Ada?check} \)  
  \( [\mu X. y?(item, amount).\text{if} \ldots \text{then } y?\text{check}.X \text{ else write KO.y?\text{check}}] \)  
- \( \{\text{iS, iF, bS, bF}\}!\text{check} \)  
  \( [(\mu X. y!\text{check}(F).X) + (\mu X. y!\text{contract}.y?(item, amount).y!\text{check}(F).X)] \)
System

\[ iF!(\text{Item, Amount}).\text{Ada? check} \quad [\mu X. y!(\text{item, amount}).y?\text{check}.X] \mid \]
\[ bF!(\text{Item, Amount}).\text{Ada? check} \quad [\mu X. y!(\text{item, amount}).y?\text{check}.X] \mid \]
\[ iS?(\text{Item, Amount}).\text{Ada? check} \quad [\mu X. y?(\text{item, amount}).\text{if} \ldots \text{then} y?\text{check}.X \text{ else write KO}.y?\text{check} ] \mid \]
\[ bS?(\text{Item, Amount}).\text{Ada? check} \quad [\mu X. y?(\text{item, amount}).\text{if} \ldots \text{then} y?\text{check}.X \text{ else write KO}.y?\text{check} ] \mid \]
\[ \{iS, iF, bS, bF\}!\text{check} \quad [(\mu X. y!\text{check}(F).X) \]
\[ +(\mu X. y!\text{contract}.y?(\text{item, amount}).y!\text{check}(F).X)] \parallel \]
\[ (\text{OK, OK}) \]
Syntax

Global types

\[ G ::= p \rightarrow \Pi: \{\ell_i(S_i).G_i\}_{i \in I} \]
\[ p \rightarrow \Pi: \{\lambda_i\}_{i \in I} \]
Syntax

Global types

\[ G ::= p \rightarrow \Pi : \{ \ell_i(S_i).G_i \}_{i \in I} \ | \]
\[ p \rightarrow \Pi : \{ \lambda_i \}_{i \in I} \ | \ end \]

Monitors

\[ M ::= p?\{ \ell_i(S_i).M_i \}_{i \in I} \ | \ \Pi!\{ \ell_i(S_i).M_i \}_{i \in I} \ | \]
\[ p?\{ \lambda_i \}_{i \in I} \ | \ \Pi!\{ \lambda_i \}_{i \in I} \ | \]
end
Syntax

Global types

\[ G ::= p \to \Pi : \{\ell_i(S_i).G_i\}_{i \in I} | \]
\[ p \to \Pi : \{\lambda_i\}_{i \in I} | \text{end} \]

Monitors

\[ M ::= p?\{\ell_i(S_i).M_i\}_{i \in I} | \Pi!\{\ell_i(S_i).M_i\}_{i \in I} | \]
\[ p?\{\lambda_i\}_{i \in I} | \Pi!\{\lambda_i\}_{i \in I} | \text{end} \]

Processes

\[ P ::= 0 | \text{op}.P | X | \mu X.P | \]
\[ c?\ell(x).P | c!\ell(e).P | \]
\[ c?(\lambda, T).P | c!(\lambda(F), T).P | \]
\[ \text{if } e \text{ then } P \text{ else } P | P + P \]
Syntax

Global types

\[ G ::= \begin{array}{l}
p \rightarrow \Pi : \{\ell_i(S_i).G_i\}_{i \in I} \\
p \rightarrow \Pi : \{\lambda_i\}_{i \in I}
\end{array} \quad | \quad \text{end} \]

Monitors

\[ M ::= \begin{array}{l}
p\{\ell_i(S_i).M_i\}_{i \in I} \\
p\{\lambda_i\}_{i \in I}
\end{array} \quad | \quad \Pi!\{\ell_i(S_i).M_i\}_{i \in I} \quad | \quad \Pi!\{\lambda_i\}_{i \in I} \quad | \quad \text{end} \]

Processes

\[ P ::= \begin{array}{l}
0 \\
op.P \\
X \\
\mu X.P \\
c?\ell(x).P \\
c!\ell(e).P \\
c?(\lambda, T).P \\
c!(\lambda(F), T).P \\
\text{if } e \text{ then } P \text{ else } P \\
P + P
\end{array} \]

Networks

\[ N ::= \begin{array}{l}
\text{new}(G) \\
M[P] \\
s:h \\
N \\
N \\
(\nu s)N
\end{array} \]
Syntax

Global types

\[ G ::= p \rightarrow \Pi : \{ \ell_i(S_i).G_i \}_{i \in I} \mid \]
\[ p \rightarrow \Pi : \{ \lambda_i \}_{i \in I} \mid \text{end} \]

Monitors

\[ \mathcal{M} ::= p?\{ \ell_i(S_i).\mathcal{M}_i \}_{i \in I} \mid \Pi!\{ \ell_i(S_i).\mathcal{M}_i \}_{i \in I} \mid \]
\[ p?\{ \lambda_i \}_{i \in I} \mid \Pi!\{ \lambda_i \}_{i \in I} \mid \text{end} \]

Processes

\[ P ::= 0 \mid \text{op}.P \mid X \mid \mu X.P \mid \]
\[ c?\ell(x).P \mid c!\ell(e).P \mid \]
\[ c?\lambda(T).P \mid c!(\lambda(F),T).P \mid \]
\[ \text{if } e \text{ then } P \text{ else } P \mid P + P \]

Networks

\[ N ::= \text{new}(G) \mid \mathcal{M}[P] \mid s:h \mid N \mid N \mid (\nu s)N \]

Systems

\[ S ::= N \parallel \sigma \]
Process Types

\[ T ::= \ ?\ell(S).T \mid !\ell(S).T \mid ?\lambda \mid !\lambda \mid T \wedge T \mid T \vee T \mid \text{end} \]
Process Types

\[
T ::= \ ?\ell(S).T \mid !\ell(S).T \mid ?\lambda \mid !\lambda \mid T \land T \mid T \lor T \mid \text{end}
\]

\[
\Gamma, X : T \vdash c?(\lambda, T).X \triangleright c:?\lambda \quad \Gamma, X : T \vdash c!(\lambda(F), T).X \triangleright c:!\lambda
\]
Process Types

\[ T ::= \ ?\ell(S).T \mid !\ell(S).T \mid ?\lambda \mid !\lambda \mid T \land T \mid T \lor T \mid \text{end} \]

\[ \Gamma, X : T \vdash c?(\lambda, T).X \triangleright c?\lambda \quad \Gamma, X : T \vdash c!(\lambda(F), T).X \triangleright c!:\lambda \]

\[ \Gamma \vdash P \triangleright c : T \quad \Gamma \vdash P \triangleright c : T \]

\[ \Gamma \vdash c?(\lambda, T).P \triangleright c?\lambda \quad \Gamma \vdash c!(\lambda(F), T).P \triangleright c!:\lambda \]
Process Types

\[ T \ ::= \ ?\ell(S).T \mid !\ell(S).T \mid ?\lambda \mid !\lambda \mid T \land T \mid T \lor T \mid \text{end} \]

\[ \Gamma, X : T \vdash c?(\lambda, T).X \triangleright c:?\lambda \quad \Gamma, X : T \vdash c!(\lambda(F), T).X \triangleright c:!\lambda \]

\[ \Gamma \vdash P \triangleright c : T \]

\[ \Gamma \vdash c?(\lambda, T).P \triangleright c:?\lambda \quad \Gamma \vdash c!(\lambda(F), T).P \triangleright c:!\lambda \]

\[ \Gamma \vdash e : \text{bool} \quad \Gamma \vdash P_1 \triangleright c : T_1 \quad \Gamma \vdash P_2 \triangleright c : T_2 \quad T_1 \lor T_2 \in T \]

\[ \Gamma \vdash \text{if } e \text{ then } P_1 \text{ else } P_2 \triangleright c : T_1 \lor T_2 \]
Process Types

\[ T ::= ?\ell(S).T \mid !\ell(S).T \mid ?\lambda \mid !\lambda \mid T \land T \mid T \lor T \mid \text{end} \]

\[ \Gamma, X : T \vdash c?(\lambda, T).X \triangleright c.?\lambda \quad \Gamma, X : T \vdash c!(\lambda(F), T).X \triangleright c!\lambda \]

\[ \Gamma \vdash P \triangleright c : T \quad \Gamma \vdash P \triangleright c : T \]

\[ \Gamma \vdash c?(\lambda, T).P \triangleright c.?\lambda \quad \Gamma \vdash c!(\lambda(F), T).P \triangleright c!\lambda \]

\[ \Gamma \vdash e : \text{bool} \quad \Gamma \vdash P_1 \triangleright c : T_1 \quad \Gamma \vdash P_2 \triangleright c : T_2 \quad T_1 \lor T_2 \in \mathcal{T} \]

\[ \Gamma \vdash \text{if } e \text{ then } P_1 \text{ else } P_2 \triangleright c : T_1 \lor T_2 \]

\[ \Gamma \vdash P_1 \triangleright c : T_1 \quad \Gamma \vdash P_2 \triangleright c : T_2 \quad T_1 \land T_2 \in \mathcal{T} \]

\[ \Gamma \vdash P_1 + P_2 \triangleright c : T_1 \land T_2 \]
Adequacy of Types for Monitors

\[ |p?\{\ell_i(S_i).M_i\}_{i \in I}| = \bigwedge_{i \in I} |p?\ell_i(S_i).M_i| \]

\[ |\Pi!\{\ell_i(S_i).M_i\}_{i \in I}| = \bigvee_{i \in I} |\Pi!\ell_i(S_i).M_i| \]

\[ |p?\{\lambda_i\}_{i \in I}| = \bigwedge_{i \in I} |p?\lambda_i| \quad |\Pi!\{\lambda_i\}_{i \in I}| = \bigvee_{i \in I} |\Pi!\lambda_i| \]

\[ |\text{end}| = \text{end} \]
Adequacy of Types for Monitors

\[ |p?\{\ell_i(S_i).M_i\}_{i \in I}| = \land_{i \in I} ?\ell_i(S_i).|M_i| \]
\[ |\Pi!\{\ell_i(S_i).M_i\}_{i \in I}| = \lor_{i \in I} !\ell_i(S_i).|M_i| \]
\[ |p?\{\lambda_i\}_{i \in I}| = \land_{i \in I} ?\lambda_i \quad |\Pi!\{\lambda_i\}_{i \in I}| = \lor_{i \in I} !\lambda_i \]
\[ |\text{end}| = \text{end} \]

\[ T \leq \text{end} \quad T_1 \land T_2 \leq T_i \quad T_i \leq T_1 \lor T_2 \quad (i = 1, 2) \]
\[ T_1 \leq T_2 \quad \text{implies} \quad !\ell(S).T_1 \leq !\ell(S).T_2 \quad ?\ell(S).T_1 \leq ?\ell(S).T_2 \]
\[ T \leq T_1 \quad \text{and} \quad T \leq T_2 \quad \text{imply} \quad T \leq T_1 \land T_2 \]
\[ T_1 \leq T \quad \text{and} \quad T_2 \leq T \quad \text{imply} \quad T_1 \lor T_2 \leq T \]
\[ (T_1 \lor T_2) \land T_3 = (T_1 \land T_3) \lor (T_2 \land T_3) \]
\[ (T_1 \land T_2) \lor T_3 = (T_1 \lor T_3) \land (T_2 \lor T_3) \]
Adequacy of Types for Monitors

\[ |p?\{\ell_i(S_i).M_i\}_{i\in I} = \bigwedge_{i\in I}?\ell_i(S_i).|M_i| \]
\[ |\Pi!\{\ell_i(S_i).M_i\}_{i\in I} = \bigvee_{i\in I}!\ell_i(S_i).|M_i| \]
\[ |p?\{\lambda_i\}_{i\in I} = \bigwedge_{i\in I}?\lambda_i \quad |\Pi!\{\lambda_i\}_{i\in I} = \bigvee_{i\in I}!\lambda_i \]
\[ |\text{end}| = \text{end} \]

\[ T \leq \text{end} \quad T_1 \land T_2 \leq T_i \quad T_i \leq T_1 \lor T_2 \quad (i = 1, 2) \]
\[ T_1 \leq T_2 \quad \text{implies} \quad !\ell(S).T_1 \leq !\ell(S).T_2 \quad ?\ell(S).T_1 \leq ?\ell(S).T_2 \]
\[ T \leq T_1 \quad \text{and} \quad T \leq T_2 \quad \text{imply} \quad T \leq T_1 \land T_2 \]
\[ T_1 \leq T \quad \text{and} \quad T_2 \leq T \quad \text{imply} \quad T_1 \lor T_2 \leq T \]
\[ (T_1 \lor T_2) \land T_3 = (T_1 \land T_3) \lor (T_2 \land T_3) \]
\[ (T_1 \land T_2) \lor T_3 = (T_1 \lor T_3) \land (T_2 \lor T_3) \]

A type \( T \) is \textbf{adequate} for a monitor \( M \) (\( T \propto M \)) if \( T \leq |M| \).
Operational Semantics

LTS for monitors

\[ p?\{\ell_i(S_i).M_i\}_{i \in I} \xrightarrow{p?\ell_i} M_j \]
\[ \Pi!\{\ell_i(S_i).M_i\}_{i \in I} \xrightarrow{\Pi!\ell_i} M_j \quad j \in I \]
\[ p?\{\lambda_i\}_{i \in I} \xrightarrow{p?\lambda_i} \]
\[ \Pi!\{\lambda_i\}_{i \in I} \xrightarrow{\Pi!\lambda_i} \quad j \in I \]
Operational Semantics

**LTS for monitors**

\[ \begin{align*}
    p?\{\ell_i(S_i).M_i\}_{i \in I} & \xrightarrow{p?\ell_i} M_j \\
    \Pi!\{\ell_i(S_i).M_i\}_{i \in I} & \xrightarrow{\Pi!\ell_i} M_j \\
    p?\{\lambda_i\}_{i \in I} & \xrightarrow{p?\lambda_i} \Pi!\{\lambda_i\}_{i \in I} \\
    j \in I
\end{align*} \]

**LTS for processes**

\[ \begin{align*}
    s[p]?\ell(x).P & \xrightarrow{s[p]?\ell(v)} P\{v/x\} \\
    s[p]!\ell(e).P & \xrightarrow{s[p]!(\ell(v))} P \quad e \downarrow v \\
    s[p]?(\lambda, T).P & \xrightarrow{s[p]?(\lambda(F), T)} P \\
    s[p]!(\lambda(F), T).P & \xrightarrow{s[p]!(\lambda(F), T)} P
\end{align*} \]
Operational Semantics

LTS for monitors
\[ p?\{\ell_i(S_i).M_i\}_{i \in I} \xrightarrow{p?\ell_i} M_j \]
\[ \Pi!\{\ell_i(S_i).M_i\}_{i \in I} \xrightarrow{\Pi!\ell_i} M_j \quad j \in I \]
\[ p?\{\lambda_i\}_{i \in I} \xrightarrow{p?\lambda_i} \Pi!\{\lambda_i\}_{i \in I} \xrightarrow{\Pi!\lambda_i} j \in I \]

LTS for processes
\[ s[p]?\ell(x).P \xrightarrow{s[p]?\ell(v)} P\{v/x\} \]
\[ s[p]!\ell(e).P \xrightarrow{s[p]!\ell(v)} P \quad e \downarrow v \]
\[ s[p]?((\lambda, T)).P \xrightarrow{s[p]?((\lambda,F),T)} P \]
\[ s[p]!(\lambda(F), T).P \xrightarrow{s[p]!((\lambda(F),T)} P \]

\[ \Pi = pa(G) \quad M_p = G \upharpoonright p \quad \forall p \in \Pi. \ (P_p, T_p) \in P \& T_p \propto M_p \]
\[ \text{new}(G) \xrightarrow{\nu s} \left( \prod_{p \in \Pi} M_p[P_p\{s[p]/y\} | s : \varnothing} \right) \]
Operational Semantics

\[ M \xrightarrow{q?\ell} M' \quad P \xrightarrow{s[p]?\ell(v)} P' \]

\[ M[P] \mid s : (q, p, \ell(v)) \cdot h \rightarrow M'[P'] \mid s : h \]
Operational Semantics

\[
\begin{align*}
\mathcal{M} \xrightarrow{q?\ell} \mathcal{M}' \\
\mathcal{M}[P] \mid s : (q, p, \ell(v)) \cdot h \longrightarrow \mathcal{M}'[P'] \mid s : h \\
\mathcal{M} \xrightarrow{\Pi!\ell} \mathcal{M}' \\
\mathcal{M}[P] \mid s : h \longrightarrow \mathcal{M}'[P'] \mid s : h \cdot (p, \Pi, \ell(v))
\end{align*}
\]
Operational Semantics

\[
\begin{align*}
\mathcal{M} & \xrightarrow{q?\ell} \mathcal{M}' & P & \xrightarrow{s[p]?\ell(v)} P' \\
\mathcal{M}[P] \mid s : (q, p, \ell(v)) \cdot h & \rightarrow \mathcal{M}'[P'] \mid s : h
\end{align*}
\]

\[
\begin{align*}
\mathcal{M} & \xrightarrow{\Pi!\ell} \mathcal{M}' & P & \xrightarrow{s[p]!\ell(v)} P' \\
\mathcal{M}[P] \mid s : h & \rightarrow \mathcal{M}'[P'] \mid s : h \cdot (p, \Pi, \ell(v))
\end{align*}
\]

\[
\begin{align*}
\mathcal{M} & \xrightarrow{q?\lambda} P & \xrightarrow{s[p]?\lambda(T)} P' & G \upharpoonright p = \mathcal{M}' & T \varpropto \mathcal{M}' \\
\mathcal{M}[P] \mid s : (q, p, \lambda(G)) \cdot h & \rightarrow \mathcal{M}'[P'] \mid s : h
\end{align*}
\]
Operational Semantics

\[
\begin{align*}
\mathcal{M} \xrightarrow{q?\ell} \mathcal{M}' & \quad P \xrightarrow{s[p]?\ell(v)} P' \\
\mathcal{M}[P] \mid s : (q, p, \ell(v)) \cdot h \rightarrow \mathcal{M}'[P'] \mid s : h
\end{align*}
\]

\[
\begin{align*}
\mathcal{M} \xrightarrow{\prod!\ell} \mathcal{M}' & \quad P \xrightarrow{s[p]!\ell(v)} P' \\
\mathcal{M}[P] \mid s : h \rightarrow \mathcal{M}'[P'] \mid s : h \cdot (p, \Pi, \ell(v))
\end{align*}
\]

\[
\begin{align*}
\mathcal{M} \xrightarrow{\prod!\lambda} P \xrightarrow{s[p]!(\lambda(F), T)} P' & \quad F(\sigma) = G \quad \mathcal{M}_p = G \upharpoonright p \quad T \propto \mathcal{M}_p \quad h \ \text{\lambda-free}
\end{align*}
\]

\[
\begin{align*}
\Pi' = \text{pa}(G) \quad \forall q \in \Pi'. \mathcal{M}_q = G \upharpoonright q \\
\forall q \in \Pi' \setminus (\Pi \cup \{p\}). \ (P_q, T_q) \in \mathcal{P} \land T_q \propto \mathcal{M}_q
\end{align*}
\]

\[
\begin{align*}
\mathcal{M}[P] \mid s : h \parallel \sigma \rightarrow \mathcal{M}_p[P'] \mid \prod_{q \in \Pi' \setminus (\Pi \cup \{p\})} \mathcal{M}_q[P_q{s[q]/y_q}] \mid \\
\quad s : h \cdot (p, \Pi, \lambda(G)) \parallel \sigma
\end{align*}
\]
a system whose network is a parallel composition of session initiators is initial
Progress

A system whose network is a parallel composition of session initiators is **initial**.

A collection $\mathcal{P}$ is **complete** if, for every global type $G$ in the domain of an adaptation function which occurs in a process belonging to $\mathcal{P}$, there are processes in $\mathcal{P}$ whose types are adequate for the monitors obtained by projecting $G$ onto its participants.
a system whose network is a parallel composition of session initiators is initial

a collection $\mathcal{P}$ is complete if, for every global type $G$ in the domain of an adaptation function which occurs in a process belonging to $\mathcal{P}$, there are processes in $\mathcal{P}$ whose types are adequate for the monitors obtained by projecting $G$ onto its participants

If $\mathcal{P}$ is complete, $S$ is an initial system and $S \xrightarrow{\mathcal{P}} S'$, then $S'$ has progress, i.e.

1. every input monitored process will always (eventually) receive a message, and
2. every message in a queue will always (eventually) be received by an input monitored process.
Focus: self-adaptiveness in the context of multiparty sessions
Focus: self-adaptiveness in the context of multiparty sessions

Main ingredients:
Focus: self-adaptiveness in the context of multiparty sessions

Main ingredients:

- **global types** representing the overall communication choreography
Summary

Focus: self-adaptiveness in the context of multiparty sessions

Main ingredients:

- **global types** representing the overall communication choreography
- **monitors** adapting the behaviour of processes to the prescriptions of global types
Summary

Focus: self-adaptiveness in the context of multiparty sessions

Main ingredients:

- **global types** representing the overall communication choreography
- **monitors** adapting the behaviour of processes to the prescriptions of global types
- a **global state** implementing the control data
Summary

Focus: self-adaptiveness in the context of multiparty sessions

Main features:
Summary

Focus: **self-adaptiveness in the context of multiparty sessions**

Main features:

- the association of a monitor with a compliant process incarnates a single participant
Summary

Focus: **self-adaptiveness** in the context of **multiparty sessions**

Main features:

- the association of a monitor with a compliant process incarnates a single participant
- the choreography is updated at runtime, in response to changing conditions in the global state
Focus: **self-adaptiveness** in the context of **multiparty sessions**

Main features:

- the association of a monitor with a compliant process incarnates a single participant
- the choreography is updated at runtime, in response to changing conditions in the global state
- a decentralised control of the adaptation: any participant can be in charge of checking global data and sending the adaptation request
Summary

Focus: **self-adaptiveness** in the context of **multiparty sessions**

Main features:

- the association of a monitor with a compliant process incarnates a single participant
- the choreography is updated at runtime, in response to changing conditions in the global state
- a decentralised control of the adaptation: any participant can be in charge of checking global data and sending the adaptation request
- the dynamic system reconfiguration can add new participants, while some of the old participants are not longer involved
Summary

Focus: self-adaptiveness in the context of multiparty sessions

Main features:

- the association of a monitor with a compliant process incarnates a single participant
- the choreography is updated at runtime, in response to changing conditions in the global state
- a decentralised control of the adaptation: any participant can be in charge of checking global data and sending the adaptation request
- the dynamic system reconfiguration can add new participants, while some of the old participants are not longer involved
- processes, that are simply implementation code, can follow different incompatible computational paths
Related Papers

- T.-C. Chen, L. Bocchi, P.-M. Deniélou, K. Honda, and N. Yoshida, “Asynchronous Distributed Monitoring for Multiparty Session Enforcement”: the monitors prescribe not only the types of the exchanged data, but also that the values of these data satisfy some predicates.
Related Papers

- T.-C. Chen, L. Bocchi, P.-M. Deniélou, K. Honda, and N. Yoshida, “Asynchronous Distributed Monitoring for Multiparty Session Enforcement”: the monitors prescribe not only the types of the exchanged data, but also that the values of these data satisfy some predicates.

- G. Anderson and J. Rathke, “Dynamic Software Update for Message Passing Programs”: global and session types are used to guarantee deadlock-freedom in a calculus of multiparty sessions with asynchronous communications.
Related Papers

- T.-C. Chen, L. Bocchi, P.-M. Deniélon, K. Honda, and N. Yoshida, “Asynchronous Distributed Monitoring for Multiparty Session Enforcement”: the monitors prescribe not only the types of the exchanged data, but also that the values of these data satisfy some predicates.

- G. Anderson and J. Rathke, “Dynamic Software Update for Message Passing Programs”: global and session types are used to guarantee deadlock-freedom in a calculus of multiparty sessions with asynchronous communications.

Future Work

- experiment with implementations of our approach, to evaluate its feasibility
Future Work

- experiment with implementations of our approach, to evaluate its feasibility
- allow the global state to contain dynamically evolving semantic information about processes, such as reputation or performance rates
Future Work

- experiment with implementations of our approach, to evaluate its feasibility
- allow the global state to contain dynamically evolving semantic information about processes, such as reputation or performance rates
- exploit a consensus algorithm for choosing a single one among all the processes matching a monitor, using rates of processes
Future Work

- experiment with implementations of our approach, to evaluate its feasibility
- allow the global state to contain dynamically evolving semantic information about processes, such as reputation or performance rates
- exploit a consensus algorithm for choosing a single one among all the processes matching a monitor, using rates of processes