Software Engineering and Service-Oriented Systems

– A Calculus for Orchestration of Web Services –

Francesco Tiezzi

IMT - Institutions, Markets, Technologies

Institute for Advanced Studies Lucca

Lucca, Italy - September, 2012

In co-operation with SENSORIA members, in particular Rosario Pugliese

Motivation

Deficiency

Current software engineering technologies for SOC

- remain at a linguistic level
- do not support analytical tools for checking that SOC applications enjoy desirable correctness properties

Goal

Develop *formal reasoning mechanisms* and *analytical tools* for checking that services (possibly resulting from a *composition*) meet desirable properties and do not manifest unexpected behaviors

Motivation

Deficiency

Current software engineering technologies for SOC

- remain at a linguistic level
- do not support analytical tools for checking that SOC applications enjoy desirable correctness properties

Goal

Develop *formal reasoning mechanisms* and *analytical tools* for checking that services (possibly resulting from a *composition*) meet desirable properties and do not manifest unexpected behaviors

Approach

Goal

Developing *formal reasoning mechanisms* and *analytical tools* for checking that the services resulting from a *composition* meet desirable correctness properties and do not manifest unexpected behaviors

Approach: rely on Process Calculi

- Convey in a distilled form the paradigm at the heart of SOC (being defined algebraically, they are inherently compositional)
- Provide linguistic formalisms for description of service-based applications and their composition
- Hand down a large set of reasoning mechanisms and analytical tools, e.g. typing systems and model checkers

Approach

Goal

Developing *formal reasoning mechanisms* and *analytical tools* for checking that the services resulting from a *composition* meet desirable correctness properties and do not manifest unexpected behaviors

Approach: rely on Process Calculi

- Convey in a distilled form the paradigm at the heart of SOC (being defined algebraically, they are inherently compositional)
- Provide linguistic formalisms for description of service-based applications and their composition
- Hand down a large set of reasoning mechanisms and analytical tools, e.g. typing systems and model checkers

Process Calculi for SOC

- To model service composition, many process calculi-like formalisms have been designed

- Most of them only consider a few specific features separately, possibly by embedding ‘ad hoc’ constructs within some well-studied process calculus
  (e.g., the variants of CSP/π-calculus with transactions)

- One major goal is assessing the adequacy of diverse sets of primitives w.r.t. modelling, combining and analysing service-oriented systems

Process Calculi for SOC: an overview

Process calculi for SOC can be classified according to the approach used for maintaining the link between *caller* and *callee*

- **Sessions**: the link is determined by a private channel that is implicitly created when the first message exchange of a conversation takes place
- **Correlations**: the link is determined by correlation values included in the exchanged messages
- **No link**: some works do not take into account this aspect e.g. $\texttt{web}\pi$, $\texttt{web}\pi_\infty$, CSP/π-calculus + transactions, ...

Process Calculi for SOC: an overview

Process calculi for SOC can be classified according to the approach used for maintaining the link between *caller* and *callee*

- **Sessions**: the link is determined by a private channel that is implicitly created when the first message exchange of a conversation takes place
- **Correlations**: the link is determined by correlation values included in the exchanged messages
- **No link**: some works do not take into account this aspect e.g. $\mathtt{web}\pi$, $\mathtt{web}\pi_\infty$, CSP/π-calculus + transactions, ...

Process Calculi for SOC: an overview

Process calculi for SOC can be classified according to the approach to maintain the link between *caller* and *callee*

- **Sessions**: the link is determined by a private channel that is implicitly created when the first message exchange of a conversation takes place
  - *dyadic*: they can be further grouped according to the inter-session communication mechanism
    - CASPIS: dataflow communication
    - SSCC: stream-based communication
    - π-calculus + sessions (in many works): session delegation
  - *multiparty*:
    - Conversation Calculus, μse,
      π-calculus + (asynchronous/synchronous) multiparty sessions
- **Correlations**: the link is determined by correlation values included in the exchanged messages
  - *stateful*: every service instance has an explicit state
    - WS-CALCULUS
    - SOCK
  - *stateless*: state is not explicitly modelled
    - COWS

Process Calculi for SOC: an overview

Process calculi for SOC can be classified according to the approach to maintain the link between *caller* and *callee*

- **Sessions**: the link is determined by a private channel that is implicitly created when the first message exchange of a conversation takes place
  - ★ *dyadic*: they can be further grouped according to the inter-session communication mechanism
    - CASPIS: dataflow communication
    - SSCC: stream-based communication
    - π-calculus + sessions (in many works): delegation
  - ★ *multiparty*:
    - Conversation Calculus, μse
      π-calculus + (asynchronous/synchronous) multiparty sessions
- **Correlations**: the link is determined by correlation values included in the exchanged messages
  - ★ *stateful*: every service instance has an explicit state
    - WS-CALCULUS
    - SOCK
  - ★ *stateless*: state is not explicitly modelled
    - COWS

Process Calculi for SOC: an overview

Process calculi for SOC can be classified according to the approach to maintain the link between *caller* and *callee*

- **Sessions**: the link is determined by a private channel that is implicitly created when the first message exchange of a conversation takes place
- **Correlations**: the link is determined by correlation values included in the exchanged messages
  - *stateful*: every service instance has an explicit state
    - WS-CALCULUS
    - SOCK
  - *stateless*: state is not explicitly modelled
    - COWS

COWS [ESOP'07]

A process calculus for specifying and combining service-oriented applications, while modelling their dynamic behaviour

An introduction to COWS

COWS: a Calculus for Orchestration of Web Services

WS-BPEL

- Inspired by
  - the OASIS standard WS-BPEL for WS orchestration
  - previous work on process calculi
- Indeed, COWS intends to be a foundational model not specifically tight to Web services' current technologies
- COWS combines in an original way a number of constructs and features borrowed from well-known process calculi

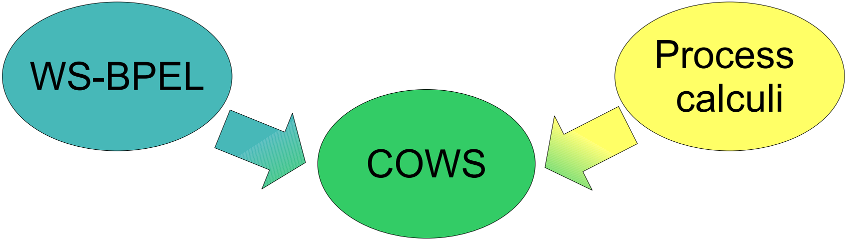
COWS: a Calculus for Orchestration of Web Services

WS-BPEL

Process calculi

- Inspired by
  - the OASIS standard WS-BPEL for WS orchestration
  - previous work on process calculi
- Indeed, COWS intends to be a foundational model not specifically tight to Web services' current technologies
- COWS combines in an original way a number of constructs and features borrowed from well-known process calculi

COWS: a Calculus for Orchestration of Web Services



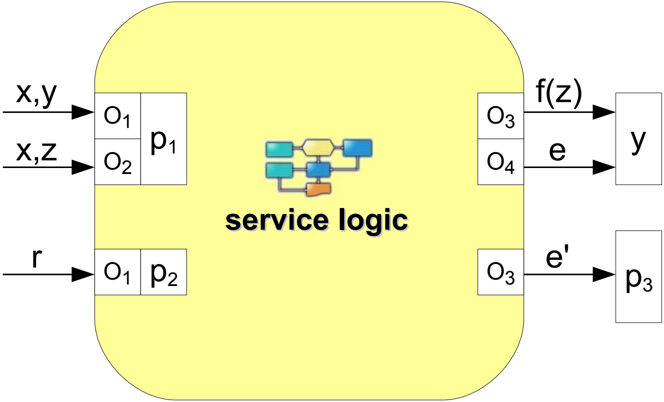
WS-BPEL

COWS

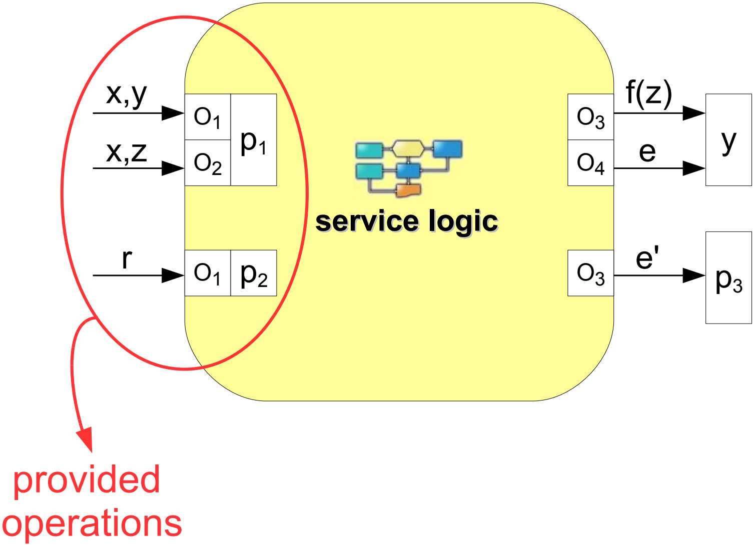
Process calculi

- Inspired by
  - the OASIS standard WS-BPEL for WS orchestration
  - previous work on process calculi
- Indeed, COWS intends to be a foundational model not specifically tight to Web services' current technologies
- COWS combines in an original way a number of constructs and features borrowed from well-known process calculi

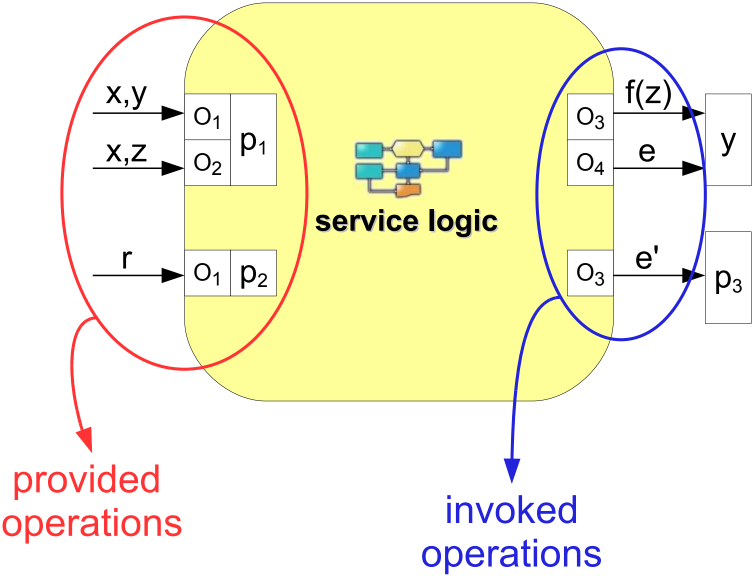
The notion of service in COWS



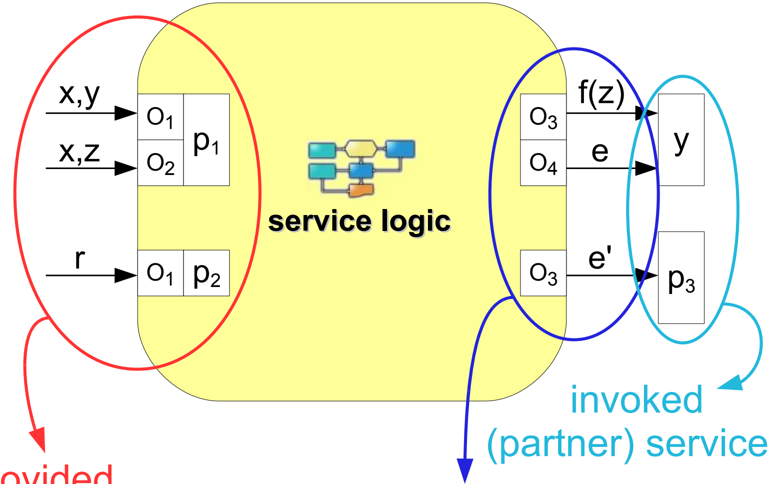
The notion of service in COWS



The notion of service in COWS



The notion of service in COWS

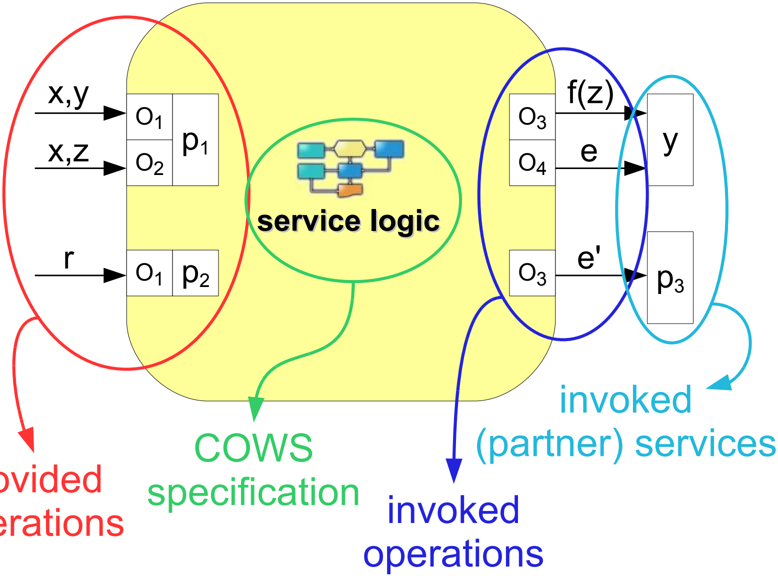


invoked (partner) services

provided operations

invoked operations

The notion of service in COWS



provided operations

COWS specification

invoked operations

invoked (partner) services

COWS in three steps

μCOWS^m (micro COWS minus priority)

Communication activities

- Invoke
- Receive

Control flow activities

- Parallel composition
- Choice
- Replication
- Delimitation

COWS in three steps

μCOWS^m (micro COWS minus priority)

Communication activities

- Invoke
- Receive

Control flow activities

- Parallel composition
- Choice
- Replication
- Delimitation

- Priority in the parallel composition

COWS in three steps

μCOWS (micro COWS)

μCOWS^m (micro COWS minus priority)

Communication activities

- Invoke
- Receive

Control flow activities

- Parallel composition
- Choice
- Replication
- Delimitation

- Priority in the parallel composition

COWS in three steps

μCOWS (micro COWS)

μCOWS^m (micro COWS minus priority)

Communication activities

- Invoke
- Receive

Control flow activities

- Parallel composition
- Choice
- Replication
- Delimitation

- Priority in the parallel composition

Termination activities

- Kill activity
- Protection

COWS in three steps

COWS (Calculus for Orchestration of Web Services)

μCOWS (micro COWS)

μCOWS^m (micro COWS minus priority)

Communication activities

- Invoke
- Receive

Control flow activities

- Parallel composition
- Choice
- Replication
- Delimitation

- Priority in the parallel composition

Termination activities

- Kill activity
- Protection

Syntax of μCOWS^m

$$\begin{array}{lll}
s ::= & & \text{(services)}\\
& u \bullet u'!\bar{\epsilon} & \quad \text{(invoke)}\\
\mid & \sum_{i=0}^{r} g_i . s_i & \quad \text{(receive-guarded choice)}\\
\mid & s \mid s & \quad \text{(parallel composition)}\\
\mid & [u]\, s & \quad \text{(delimitation)}\\
\mid & *\, s & \quad \text{(replication)}\\
g ::= & & \text{(guards)}\\
& p \bullet o?\bar{w} & \quad \text{(receive)}
\end{array}$$

(notations)
ϵ: *expressions*
x: *variables*
v: *values*
n, p, o: *names*
u: *variables* | *names*
w: *variables* | *values*

μCOWS^m vs. π-calculus, fusion, Value-passing CCS, Dπ, ...

$$\left.\begin{array}{l}
\bullet\ \ \text{asynchronous and polyadic communication}\\
\bullet\ \ \text{input} - \text{guarded choice}\\
\bullet\ \ \text{polyadic synchronization}\\
\bullet\ \ \text{localised channels}
\end{array}\right\}\ \pi\text{-calculus}$$

$$\bullet\ \ \text{global scoping (and non} - \text{binding input)}\ \ \}\ \text{fusion}$$

$$\bullet\ \ \text{distinction between variables and values}\ \ \}\ \text{vp CCS},\ \ \text{App. } \pi\text{-calculus},\ \ \text{D}\pi$$

$$\bullet\ \ \text{pattern} - \text{matching}\ \ \}\ \text{Klaim}$$

Syntax of μCOWS^m

$s ::=$		(services)
	$u \bullet u'!\bar{\epsilon}$	(invoke)
$\mid$	$\sum_{i=0}^{r} g_i . s_i$	(receive-guarded choice)
$\mid$	$s \mid s$	(parallel composition)
$\mid$	$[u]\, s$	(delimitation)
$\mid$	$* \, s$	(replication)
$g ::=$		(guards)
	$p \bullet o?\bar{w}$	(receive)

(notations)

ϵ: *expressions*

x: *variables*

v: *values*

n, p, o: *names*

u: *variables* | *names*

w: *variables* | *values*

Notations

- The exact syntax of expressions is deliberately omitted
- ¯ denotes tuples of objects, e.g. $\bar{w}$ is a tuple of variables and/or values

Syntax of μCOWS^m

$s ::=$	(services)
$u \bullet u'!\bar{\epsilon}$	(invoke)
$\mid \sum_{i=0}^{r} g_i . s_i$	(receive-guarded choice)
$\mid s \mid s$	(parallel composition)
$\mid [u]\, s$	(delimitation)
$\mid * s$	(replication)
$g ::=$	(guards)
$p \bullet o?\bar{w}$	(receive)

(notations)
ϵ: *expressions*
x: *variables*
v: *values*
n, p, o: *names*
u: *variables* | *names*
w: *variables* | *values*

Communication activities

- Services are provided and invoked through communication *endpoints*, written as $p \bullet o$ (i.e. ‘partner name’ plus ‘operation name’)
- Receive activities bind neither names nor variables
- Communication is regulated by *pattern-matching*
- Partner names and operation names can be exchanged when communicating (only the ‘send capability’ is passed over)
- Communication is asynchronous

Syntax of μCOWS^m

$s ::=$		(services)
	$u \bullet u'!\bar{\epsilon}$	(invoke)
$\mid$	$\sum_{i=0}^{r} g_i . s_i$	(receive-guarded choice)
$\mid$	$s \mid s$	(parallel composition)
$\mid$	$[u]\, s$	(delimitation)
$\mid$	$* \, s$	(replication)
$g ::=$		(guards)
	$p \bullet o?\bar{w}$	(receive)

(notations)

ϵ: *expressions*

x: *variables*

v: *values*

n, p, o: *names*

u: *variables* | *names*

w: *variables* | *values*

Choice

- $+$ abbreviates binary choice, while empty choice will be denoted by **0**

Syntax of μCOWS^m

$s ::=$		(services)
	$u \bullet u' ! \bar{\epsilon}$	(invoke)
$\mid$	$\sum_{i=0}^{r} g_i . s_i$	(receive-guarded choice)
$\mid$	$s \mid s$	(parallel composition)
$\mid$	$[u]\, s$	(delimitation)
$\mid$	$* \, s$	(replication)
$g ::=$		(guards)
	$p \bullet o ? \bar{w}$	(receive)

(notations)
ϵ: *expressions*
x: *variables*
v: *values*
n, p, o: *names*
u: *variables* | *names*
w: *variables* | *values*

Parallel composition

- Permits interleaving executions of activities

Syntax of μCOWS^m

$$\begin{array}{lll}
s ::= & & \text{(services)} \\
 & u \bullet u' ! \bar{\epsilon} & \text{(invoke)} \\
 | & \sum_{i=0}^{r} g_i . s_i & \text{(receive-guarded choice)} \\
 | & s \mid s & \text{(parallel composition)} \\
 | & [u]\, s & \text{(delimitation)} \\
 | & * \, s & \text{(replication)} \\
g ::= & & \text{(guards)} \\
 & p \bullet o ? \bar{w} & \text{(receive)}
\end{array}$$

(notations)

ϵ: *expressions*

x: *variables*

v: *values*

n, p, o: *names*

u: *variables* | *names*

w: *variables* | *values*

Delimitation

- Only one binding construct: $[u]\, s$ binds u in the scope s
  - free/bound names and variables and closed terms defined accordingly
- Delimitation is used to:
  1. regulate the range of application of substitutions
  2. generate fresh names

Syntax of μCOWS^m

s ::= (services)

$u \bullet u' ! \bar{\epsilon}$ (invoke)

| $\sum_{i=0}^{r} g_i . s_i$ (receive-guarded choice)

| $s \mid s$ (parallel composition)

| $[u]\, s$ (delimitation)

| $* s$ (replication)

g ::= (guards)

$p \bullet o ? \bar{w}$ (receive)

(notations)

ϵ: *expressions*

x: *variables*

v: *values*

n, p, o: *names*

u: *variables* | *names*

w: *variables* | *values*

Replication

- Permits implementing persistent services and recursive behaviours

μCOWS^m operational semantics

Labelled transition relation $\xrightarrow{\alpha}$

Label α is generated by the following grammar:

$$\alpha \quad ::= \quad \mathrm{n} \lhd \bar{v} \quad | \quad \mathrm{n} \rhd \bar{w} \quad | \quad \sigma$$

where σ is a *substitution*

i.e. a function from variables to values (written as collections of pairs $x \mapsto v$)

and n denotes endpoints (i.e. $p \bullet o$)

Structural congruence $\equiv$

Standard laws for $\sum$, $|$ and $*$, plus:

- $[u]\,\mathbf{0} \;\equiv\; \mathbf{0}$
- $[u_1]\,[u_2]\,s \;\equiv\; [u_2]\,[u_1]\,s$
- $s_1 \mid [u]\,s_2 \;\equiv\; [u]\,(s_1 \mid s_2)$ if $u \notin \mathrm{fu}(s_1)$

  $\mathrm{fu}(s)$ denotes the set of elements occurring free in s

μCOWS^m operational semantics

Labelled transition relation $\xrightarrow{\alpha}$

Label α is generated by the following grammar:

$$\alpha \quad ::= \quad \mathrm{n} \lhd \bar{v} \quad | \quad \mathrm{n} \rhd \bar{w} \quad | \quad \sigma$$

where σ is a *substitution*

i.e. a function from variables to values (written as collections of pairs $x \mapsto v$)

and n denotes endpoints (i.e. $p \bullet o$)

Structural congruence $\equiv$

Standard laws for $\sum$, $|$ and $*$, plus:

- $[u]\,\mathbf{0} \;\equiv\; \mathbf{0}$
- $[u_1]\,[u_2]\,s \;\equiv\; [u_2]\,[u_1]\,s$
- $s_1 \mid [u]\,s_2 \;\equiv\; [u]\,(s_1 \mid s_2)$ if $u \notin \mathrm{fu}(s_1)$

$\mathrm{fu}(s)$ denotes the set of elements occurring free in s

μCOWS^m: Invoke/receive activities & Choice

Invoke activities

- Can proceed only if the expressions in the argument can be evaluated
- *Evaluation function* $[\![\_]\!]$: takes closed expressions and returns values

$$\frac{[\![\bar{\epsilon}]\!] = \bar{v}}{\mathrm{n}!\bar{\epsilon} \xrightarrow{\mathrm{n} \lhd \bar{v}} \mathbf{0}}$$

Choice (among receive activities)

- Offers an alternative choice of endpoints
- It is *not* a binder for names and variables (delimitation is used to delimit their scope)

$$\textstyle\sum_{i=1}^{r} \mathrm{n}_i?\bar{w}_i.s_i \xrightarrow{\mathrm{n}_j \rhd \bar{w}_j} s_j \qquad (1 \leq j \leq r)$$

μCOWS^m: Parallel composition

- Communication takes place when two parallel services perform matching receive and invoke activities

$$\frac{s_1 \xrightarrow{\mathrm{n} \triangleright \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \triangleleft \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma}{s_1 \mid s_2 \xrightarrow{\sigma} s_1' \mid s_2'}$$

- Execution of parallel services is interleaved

$$\frac{s_1 \xrightarrow{\alpha} s_1'}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

Matching function

$$\mathcal{M}(x, v) = \{x \mapsto v\} \qquad \mathcal{M}(v, v) = \emptyset \qquad \mathcal{M}(\langle\rangle, \langle\rangle) = \emptyset \qquad \frac{\mathcal{M}(w_1, v_1) = \sigma_1 \quad \mathcal{M}(\bar{w}_2, \bar{v}_2) = \sigma_2}{\mathcal{M}((w_1, \bar{w}_2), (v_1, \bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

μCOWS^m: Parallel composition

- Communication takes place when two parallel services perform matching receive and invoke activities

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma}{s_1 \mid s_2 \xrightarrow{\sigma} s_1' \mid s_2'}$$

- Execution of parallel services is interleaved

$$\frac{s_1 \xrightarrow{\alpha} s_1'}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

Matching function

$$\mathcal{M}(x, v) = \{x \mapsto v\} \qquad \begin{array}{c} \mathcal{M}(v, v) = \emptyset \\ \mathcal{M}(\langle\rangle, \langle\rangle) = \emptyset \end{array} \qquad \frac{\mathcal{M}(w_1, v_1) = \sigma_1 \quad \mathcal{M}(\bar{w}_2, \bar{v}_2) = \sigma_2}{\mathcal{M}((w_1, \bar{w}_2), (v_1, \bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

μCOWS^m: Parallel composition

- Communication takes place when two parallel services perform matching receive and invoke activities

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma}{s_1 \mid s_2 \xrightarrow{\sigma} s_1' \mid s_2'}$$

- Execution of parallel services is interleaved

$$\frac{s_1 \xrightarrow{\alpha} s_1'}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

Matching function

$$\mathcal{M}(x, v) = \{x \mapsto v\} \qquad \mathcal{M}(v, v) = \emptyset \qquad \mathcal{M}(\langle\rangle, \langle\rangle) = \emptyset \qquad \frac{\mathcal{M}(w_1, v_1) = \sigma_1 \quad \mathcal{M}(\bar{w}_2, \bar{v}_2) = \sigma_2}{\mathcal{M}((w_1, \bar{w}_2), (v_1, \bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

μCOWS^m: Delimitation

- $[u]\,s$ behaves like s, except when the transition label α contains u
- When the whole scope of a variable x is determined, and a communication involving x within that scope is taking place the delimitation is removed and the substitution for x is performed

$$\frac{s \xrightarrow{\alpha} s' \quad u \notin \mathrm{u}(\alpha)}{[u]\,s \xrightarrow{\alpha} [u]\,s'} \qquad \frac{s \xrightarrow{\sigma \uplus \{x \mapsto v\}} s'}{[x]\,s \xrightarrow{\sigma} s' \cdot \{x \mapsto v\}}$$

Substitutions (ranged over by σ):

- functions from variables to values (written as collections of pairs $x \mapsto v$)
- $\sigma_1 \uplus \sigma_2$ denotes the union of σ_1 and σ_2 when they have disjoint domains

$\mathrm{u}(\alpha)$ avoids capturing endpoints of actual communications,
it denotes the set of elements occurring in α,

μCOWS^m operational semantics

Labelled transition rules

$$\frac{[\![\bar{\epsilon}]\!] = \bar{v}}{\mathrm{n}!\bar{\epsilon} \xrightarrow{\mathrm{n} \lhd \bar{v}} \mathbf{0}}$$

$$\frac{1 \leq j \leq r}{\sum_{i=1}^{r} \mathrm{n}_i?\bar{w}_i.s_i \xrightarrow{\mathrm{n}_j \rhd \bar{w}_j} s_j}$$

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma}{s_1 \mid s_2 \xrightarrow{\sigma} s_1' \mid s_2'}$$

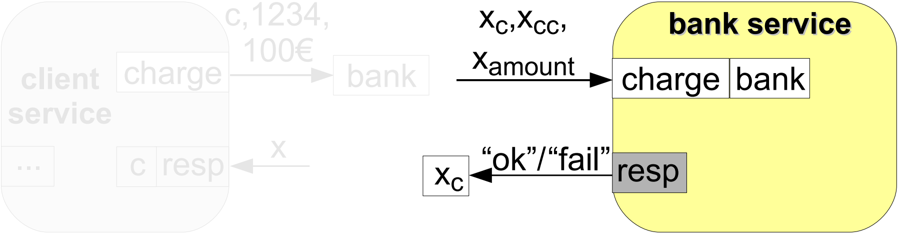
$$\frac{s_1 \xrightarrow{\alpha} s_1'}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

$$\frac{s \xrightarrow{\sigma \uplus \{x \mapsto v\}} s'}{[x]\, s \xrightarrow{\sigma} s' \cdot \{x \mapsto v\}}$$

$$\frac{s \xrightarrow{\alpha} s' \qquad u \notin \mathrm{u}(\alpha)}{[u]\, s \xrightarrow{\alpha} [u]\, s'}$$

$$\frac{s \equiv \xrightarrow{\alpha} \equiv s'}{s \xrightarrow{\alpha} s'}$$

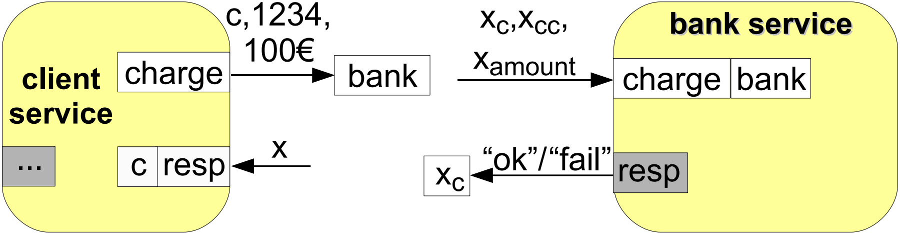
μCOWS^m: *simple* bank service example



$\mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, 1234, 100€\rangle$
$\mid [\mathsf{x}]\,(\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}\rangle.\mathsf{s} \mid \mathsf{s}')$

$[\mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}]$
$\mathsf{bank} \bullet \mathsf{charge}?\langle \mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}\rangle.$
$\mathsf{x_c} \bullet \mathsf{resp}!\langle \mathit{chk}(x_{cc}, x_{amount})\rangle$

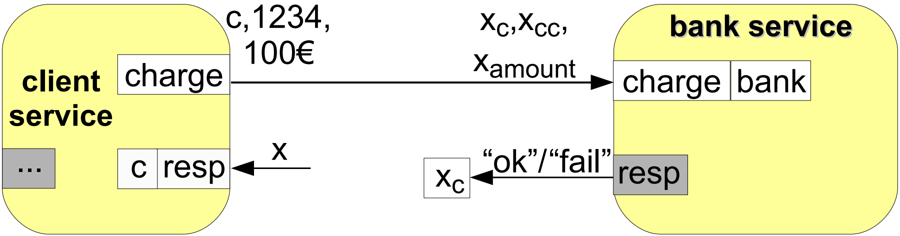
μCOWS^m: *simple* bank service example



$$\mathrm{bank} \bullet \mathrm{charge}!\langle \mathrm{c}, 1234, 100€\rangle \mid [\mathrm{x}]\,(\mathrm{c} \bullet \mathrm{resp}?\langle \mathrm{x}\rangle.\mathrm{s} \mid \mathrm{s}')$$

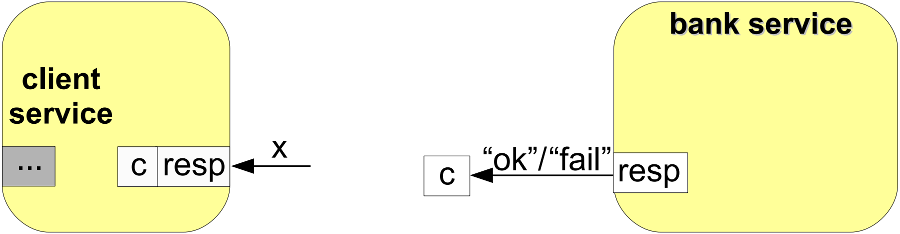
$$[\mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}]\ \mathrm{bank} \bullet \mathrm{charge}?\langle \mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}\rangle.\ \mathrm{x_c} \bullet \mathrm{resp}!\langle chk(x_{cc}, x_{amount})\rangle$$

μCOWS^m: *simple* bank service example



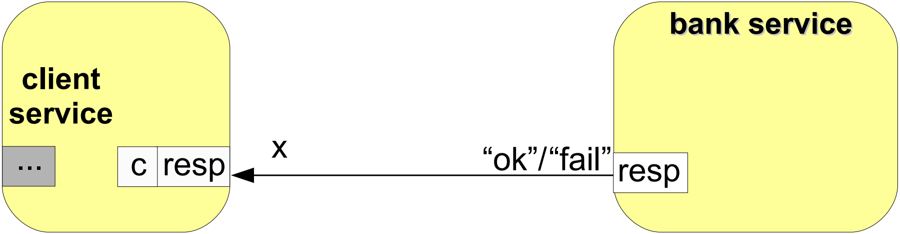
$$\begin{array}{lll} \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, 1234, 100€\rangle & & [\mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}] \\ \mid [\mathsf{x}]\,(\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}\rangle.\mathsf{s} \mid \mathsf{s}') & \mid & \mathsf{bank} \bullet \mathsf{charge}?\langle \mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}\rangle. \\ & & \mathsf{x_c} \bullet \mathsf{resp}!\langle chk(x_{cc}, x_{amount})\rangle \end{array}$$

μCOWS^m: *simple* bank service example



$[x]\,(c \bullet resp?\langle x\rangle.s \mid s') \qquad \mid \qquad c \bullet resp!\langle \mathit{chk}(1234, 100€)\rangle$

μCOWS^m: *simple* bank service example

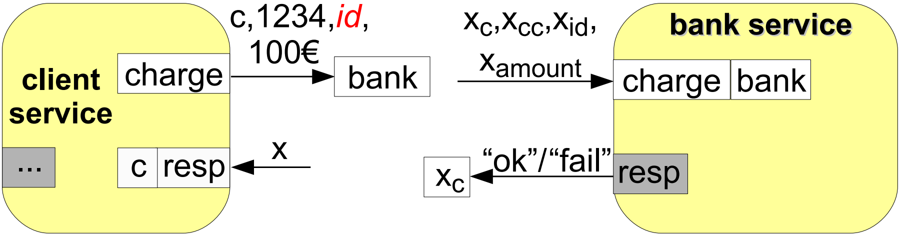


$[x]\,(c \bullet resp?\langle x\rangle.s \mid s') \qquad\qquad \mid \qquad\qquad c \bullet resp!\langle \mathit{chk}(1234, 100€)\rangle$

μCOWS^m: *simple* bank service example

$(\mathsf{s} \mid \mathsf{s}') \cdot \{\mathsf{x} \mapsto \text{“ok”} \,/\, \text{“fail”}\} \qquad \mid \qquad \mathbf{0}$

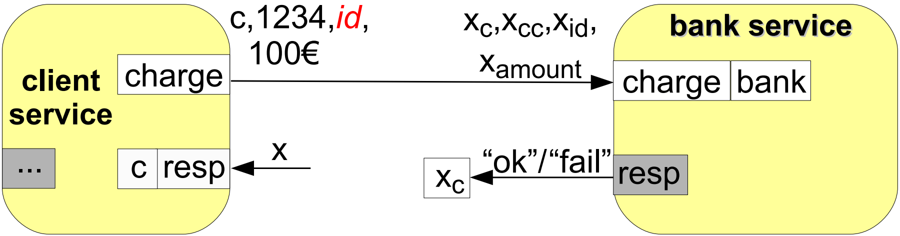
μCOWS^m: communication of private names



[id]
(bank • charge!⟨c, 1234, id, 100€⟩
| [x] (c • resp?⟨x⟩.s | s′))

[x_c, x_{cc}, x_{id}, x_{amount}]
bank • charge?⟨x_c, x_{cc}, x_{id}, x_{amount}⟩.
x_c • resp!⟨$chk(x_{cc}, x_{id}, x_{amount})$⟩

μCOWS^m: communication of private names

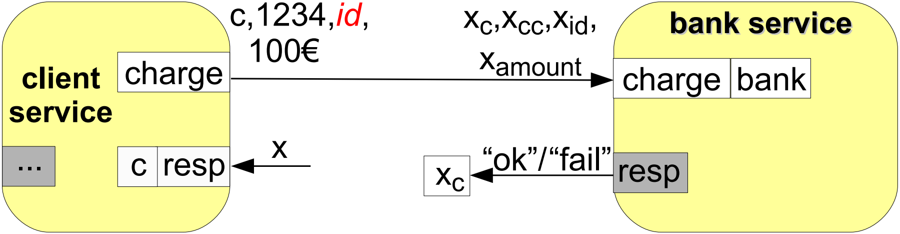


$\color{red}{[id]}$
$\color{red}{(}\mathrm{bank} \bullet \mathrm{charge}!\langle \mathrm{c}, 1234, \color{red}{\mathrm{id}}, 100€\rangle \mid [\mathrm{x}]\,(\mathrm{c} \bullet \mathrm{resp}?\langle \mathrm{x}\rangle.\mathrm{s} \mid \mathrm{s}')\ \color{red}{)}$

$\mid$

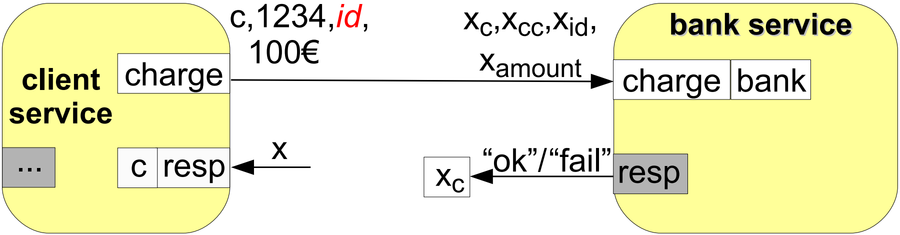
$[\mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{id}}, \mathrm{x_{amount}}]$
$\mathrm{bank} \bullet \mathrm{charge}?\langle \mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{id}}, \mathrm{x_{amount}}\rangle.$
$\mathrm{x_c} \bullet \mathrm{resp}!\langle chk(x_{cc}, x_{id}, x_{amount})\rangle$

μCOWS^m: communication of private names



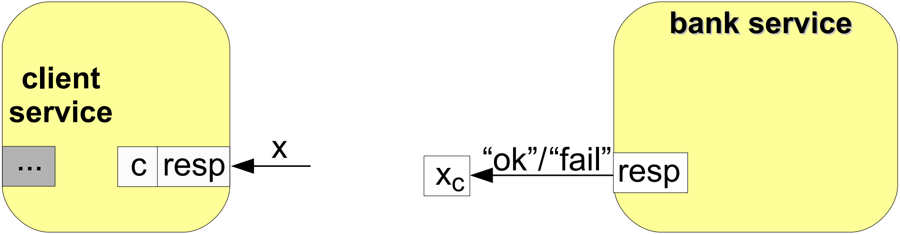
$\equiv$

μCOWS^m: communication of private names



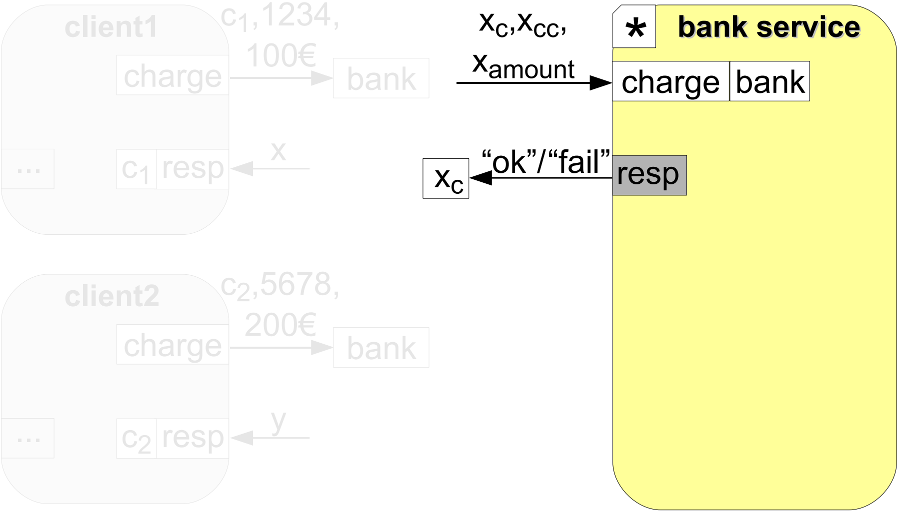
$$[id, x_c, x_{cc}, x_{id}, x_{amount}]\left(\begin{pmatrix} \mathrm{bank} \bullet \mathrm{charge}!\langle c, 1234, id, 100\text{€}\rangle \\ \mid [x]\,(c \bullet \mathrm{resp}?\langle x\rangle.s \mid s') \end{pmatrix} \mid \begin{pmatrix} \mathrm{bank} \bullet \mathrm{charge}?\langle x_c, x_{cc}, x_{id}, x_{amount}\rangle. \\ x_c \bullet \mathrm{resp}!\langle chk(x_{cc}, x_{id}, x_{amount})\rangle \end{pmatrix}\right)$$

μCOWS^m: communication of private names



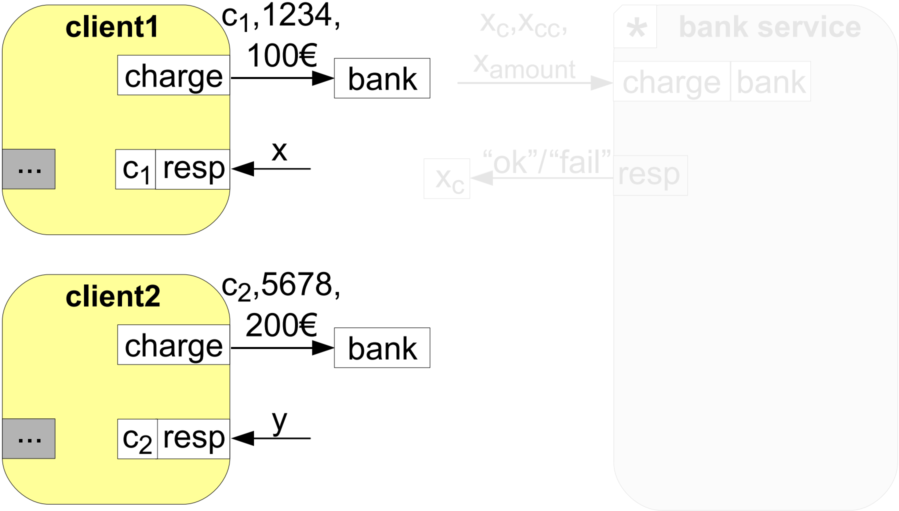
$$[id]\;(\;[x]\,(c \bullet resp?\langle x\rangle.s \mid s') \quad\mid\quad c \bullet resp!\langle chk(1234, id, 100€)\rangle\;)$$

μCOWS^m: *persistent* bank service example



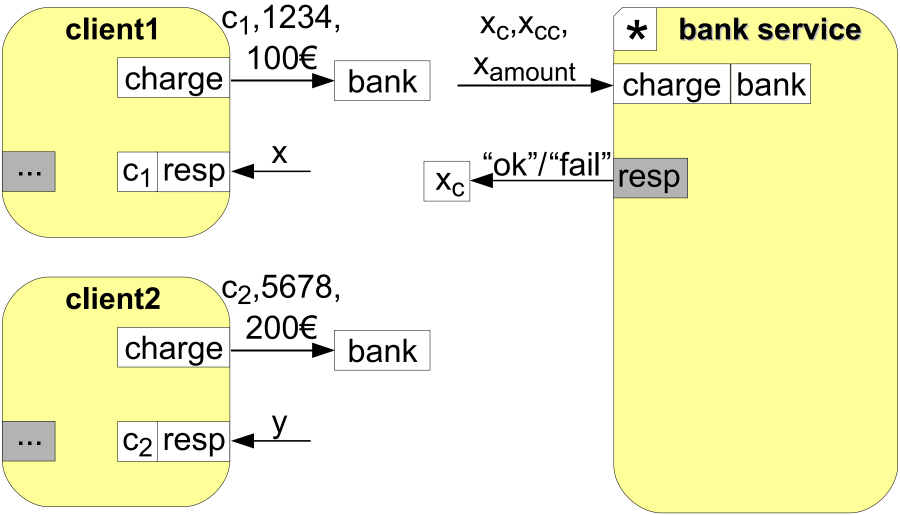
$$* \,[\mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}]\, \mathrm{bank} \bullet \mathrm{charge}?\langle \mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}\rangle.\mathrm{x_c} \bullet \mathrm{resp}!\langle chk(x_{cc}, x_{amount})\rangle$$

μCOWS^m: *persistent* bank service example

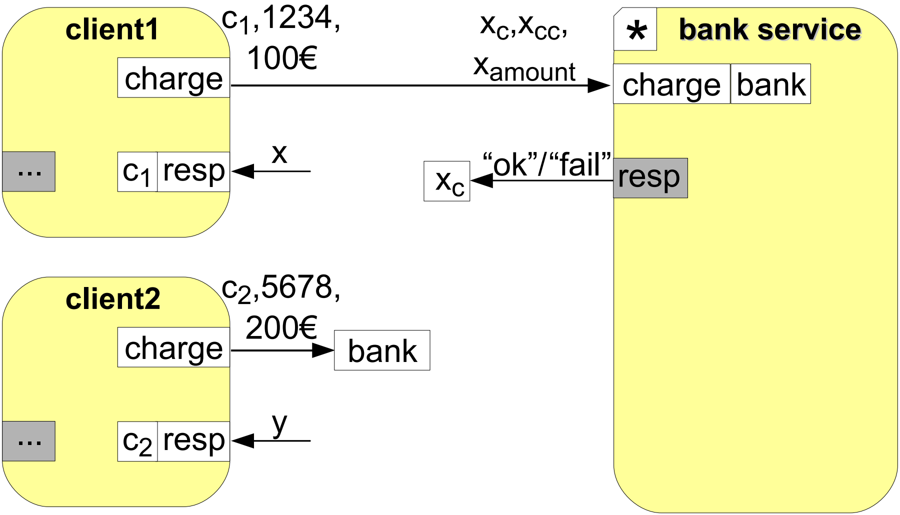


$$\begin{aligned}&\mathrm{bank} \bullet \mathrm{charge}!\langle c_1, 1234, 100€\rangle \mid [x]\, c_1 \bullet \mathrm{resp}?\langle x\rangle.s_1\\ \mid\ &\mathrm{bank} \bullet \mathrm{charge}!\langle c_2, 5678, 200€\rangle \mid [y]\, c_2 \bullet \mathrm{resp}?\langle y\rangle.s_2\end{aligned}$$

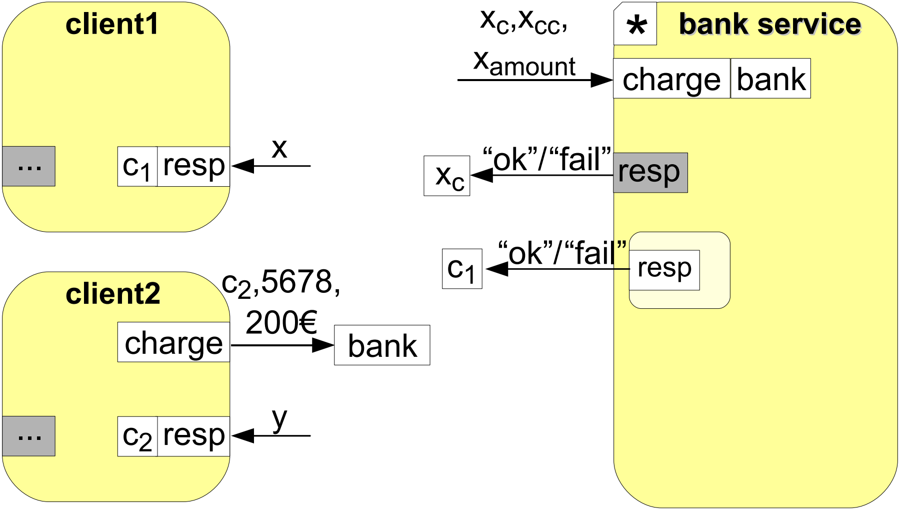
μCOWS^m: *persistent* bank service example



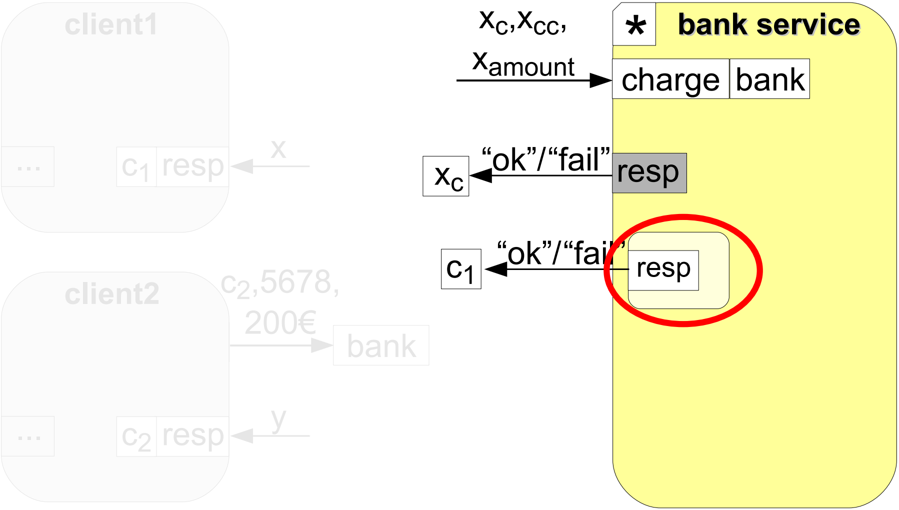
μCOWS^m: *persistent* bank service example



μCOWS^m: *persistent* bank service example



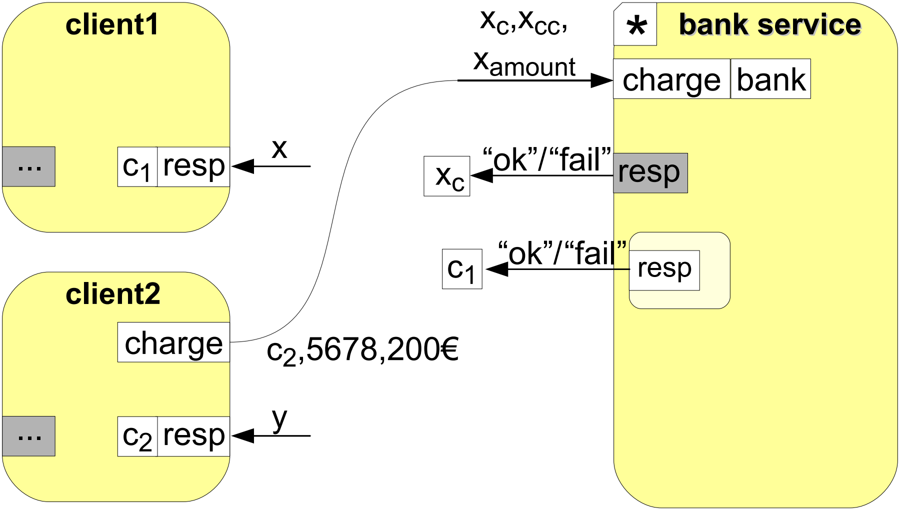
μCOWS^m: *persistent* bank service example



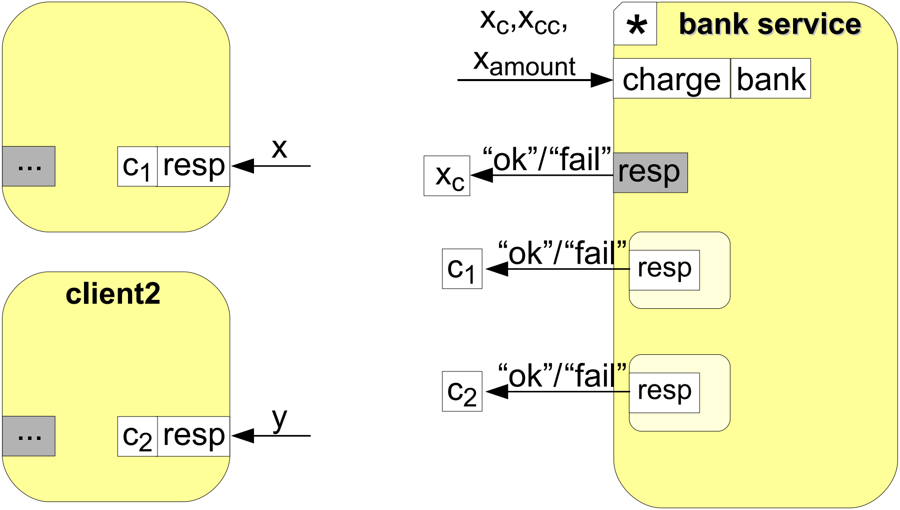
$* [x_c, x_{cc}, x_{amount}]\, \mathrm{bank} \bullet \mathrm{charge}?\langle x_c, x_{cc}, x_{amount}\rangle . x_c \bullet \mathrm{resp}!\langle \mathit{chk}(x_{cc}, x_{amount})\rangle$

$\mid\ c_1 \bullet \mathrm{resp}!\langle \mathit{chk}(1234, 100€)\rangle$

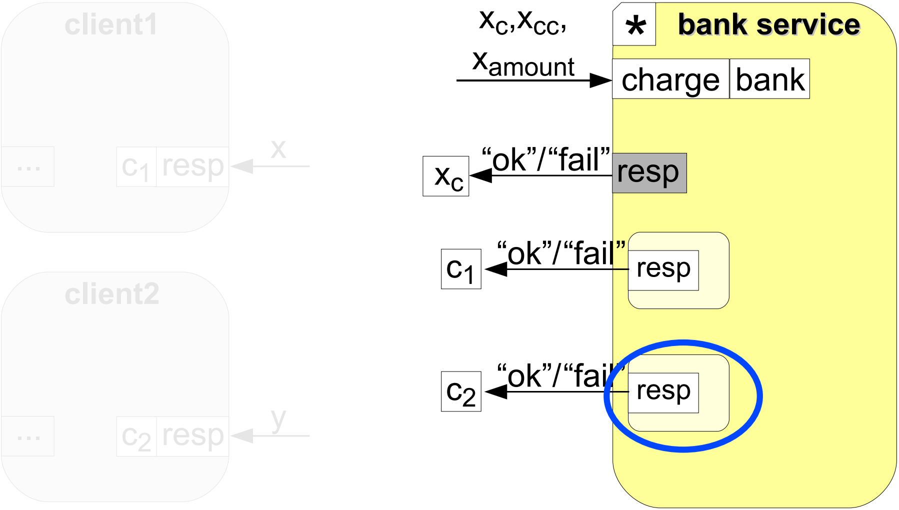
μCOWS^m: *persistent* bank service example



μCOWS^m: *persistent* bank service example



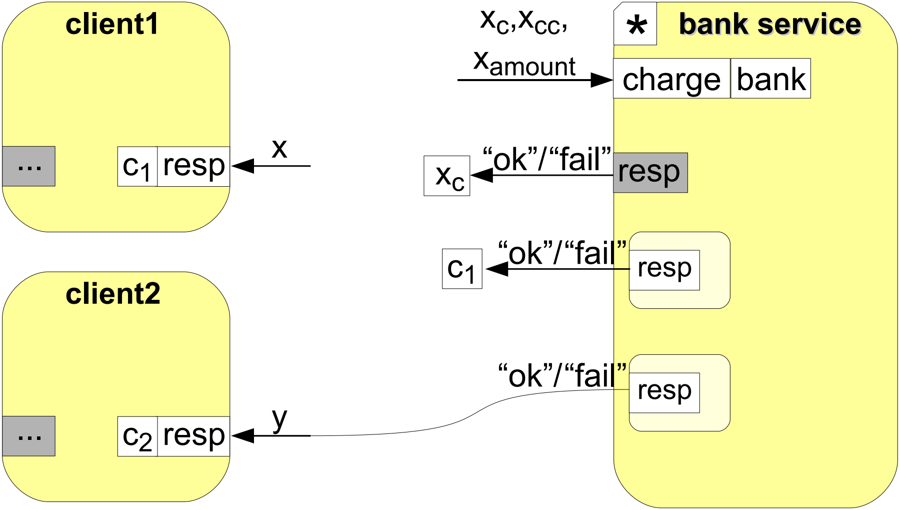
μCOWS^m: *persistent* bank service example



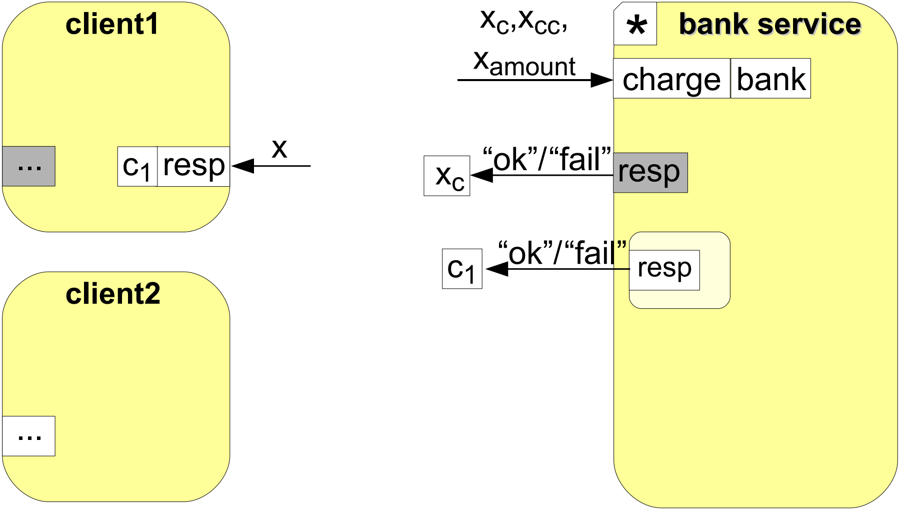
$$* \,[\mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}]\, \mathrm{bank} \bullet \mathrm{charge}?\langle \mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}\rangle . \mathrm{x_c} \bullet \mathrm{resp}!\langle chk(x_{cc}, x_{amount})\rangle$$

$$\mid\; \mathrm{c_1} \bullet \mathrm{resp}!\langle chk(1234, 100\text{€})\rangle \;\mid\; \mathrm{c_2} \bullet \mathrm{resp}!\langle chk(5678, 200\text{€})\rangle$$

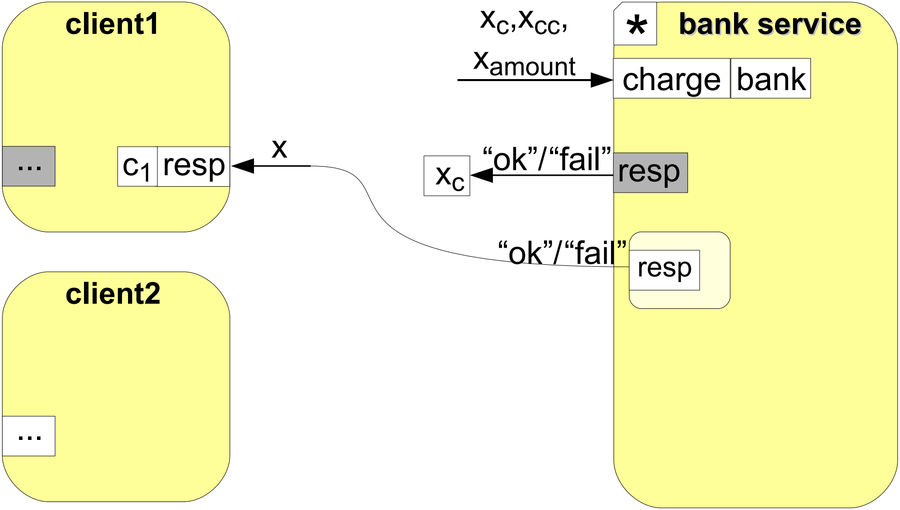
μCOWS^m: *persistent* bank service example



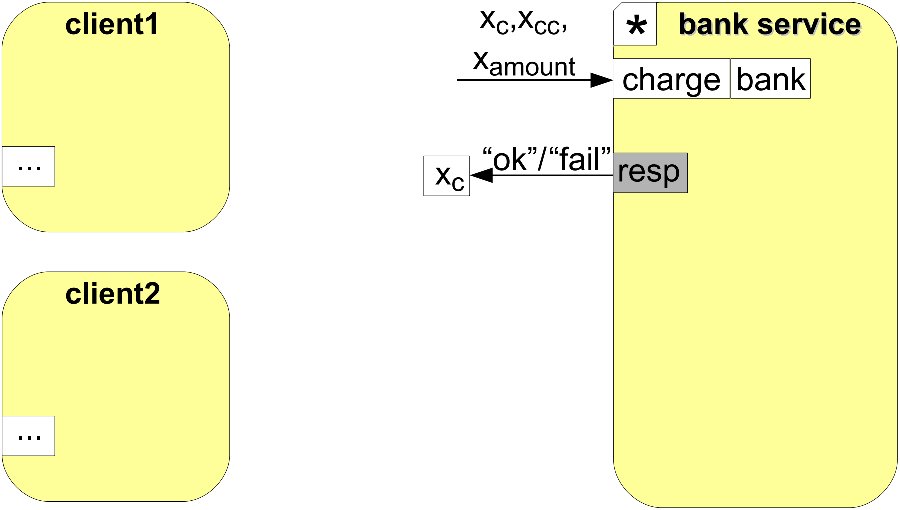
μCOWS^m: *persistent* bank service example



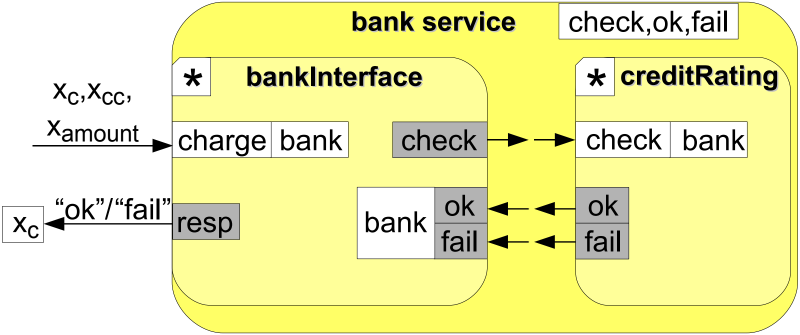
μCOWS^m: *persistent* bank service example



μCOWS^m: *persistent* bank service example

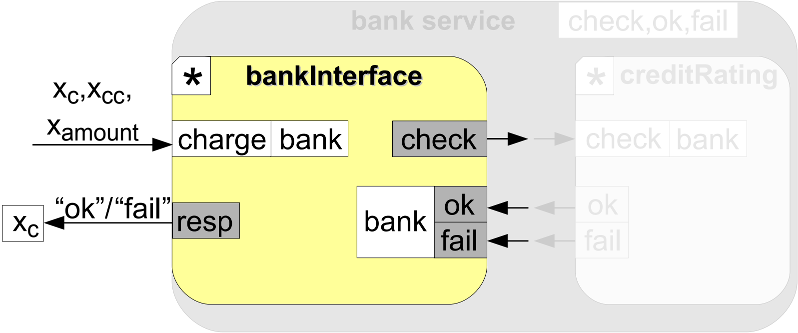


μCOWS^m: *compound* bank service example



$$[\text{check}, \text{ok}, \text{fail}]\,(\, *\,\text{bankInterface} \mid *\,\text{creditRating}\,)$$

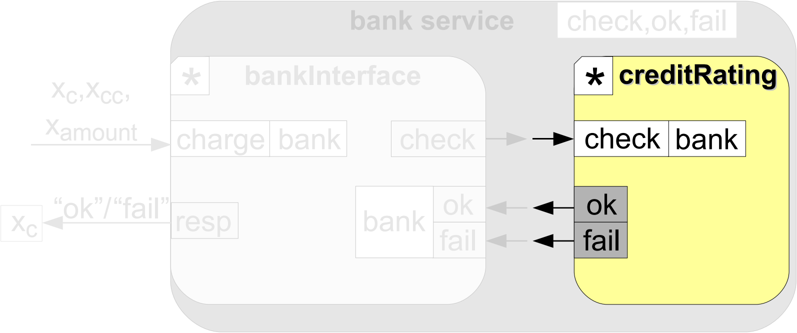
μCOWS^m: *compound* bank service example



$$[\mathsf{check}, \mathsf{ok}, \mathsf{fail}]\,(\,*\,\mathsf{bankInterface} \mid *\,\mathsf{creditRating}\,)$$

$$\begin{array}{lll}\mathsf{bankInterface} & \triangleq & [x_c, x_{cc}, x_{amount}] \\ & & \mathsf{bank} \bullet \mathsf{charge}?\langle x_c, x_{cc}, x_{amount}\rangle . \\ & & (\,\mathsf{bank} \bullet \mathsf{check}!\langle x_{cc}, x_{amount}\rangle \\ & & \;\mid \mathsf{bank} \bullet \mathsf{ok}?\langle x_{cc}\rangle .\, x_c \bullet \mathsf{resp}!\langle \text{“ok”}\rangle \\ & & \quad + \mathsf{bank} \bullet \mathsf{fail}?\langle x_{cc}\rangle .\, x_c \bullet \mathsf{resp}!\langle \text{“fail”}\rangle\,)\end{array}$$

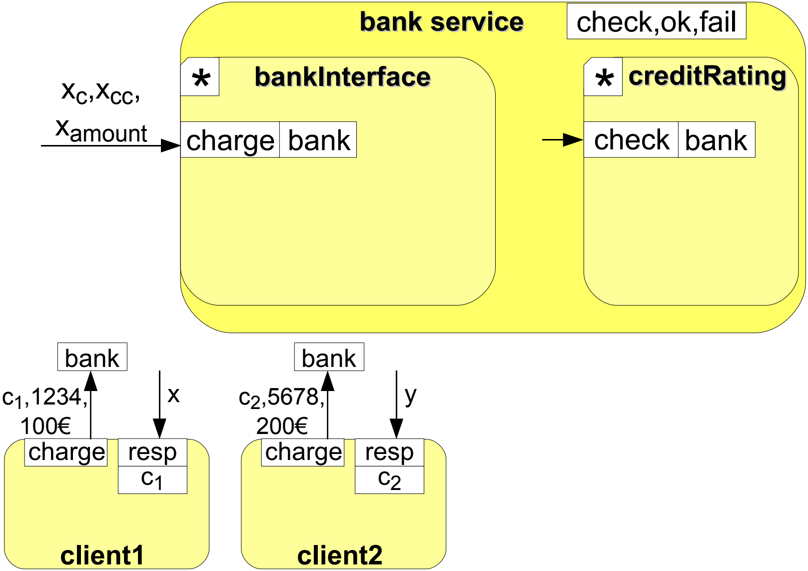
μCOWS^m: *compound* bank service example



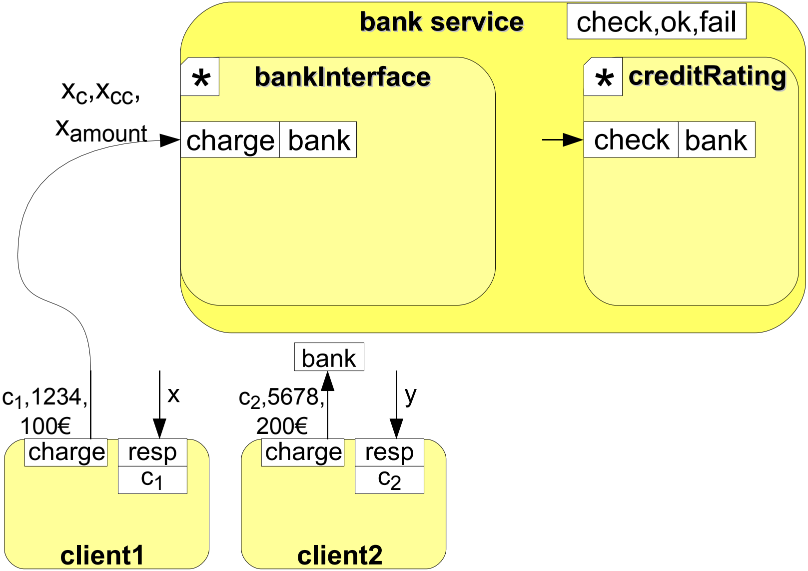
$$[\mathsf{check}, \mathsf{ok}, \mathsf{fail}]\,(\,*\,\mathsf{bankInterface} \mid *\,\mathsf{creditRating}\,)$$

$$\begin{array}{lcl} \mathsf{creditRating} & \triangleq & [\mathsf{x_{cc}}, \mathsf{x_a}] \\ & & \mathsf{bank} \bullet \mathsf{check}?\langle \mathsf{x_{cc}}, \mathsf{x_a} \rangle . \\ & & [\mathsf{p}, \mathsf{o}]\,(\,\mathsf{p} \bullet \mathsf{o}!\langle\rangle \mid \mathsf{p} \bullet \mathsf{o}?\langle\rangle .\, \mathsf{bank} \bullet \mathsf{ok}!\langle \mathsf{x_{cc}} \rangle \\ & & \qquad\qquad\qquad + \mathsf{p} \bullet \mathsf{o}?\langle\rangle .\, \mathsf{bank} \bullet \mathsf{fail}!\langle \mathsf{x_{cc}} \rangle\,) \end{array}$$

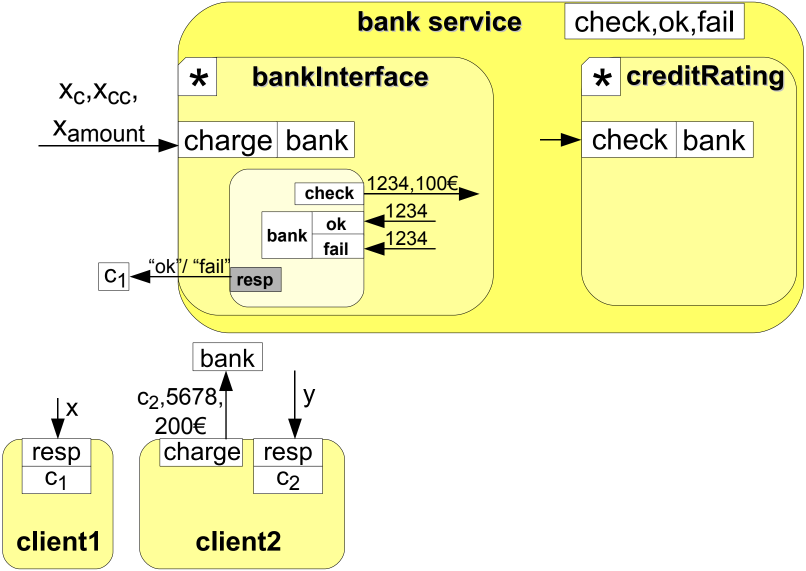
μCOWS^m: *compound* bank service example



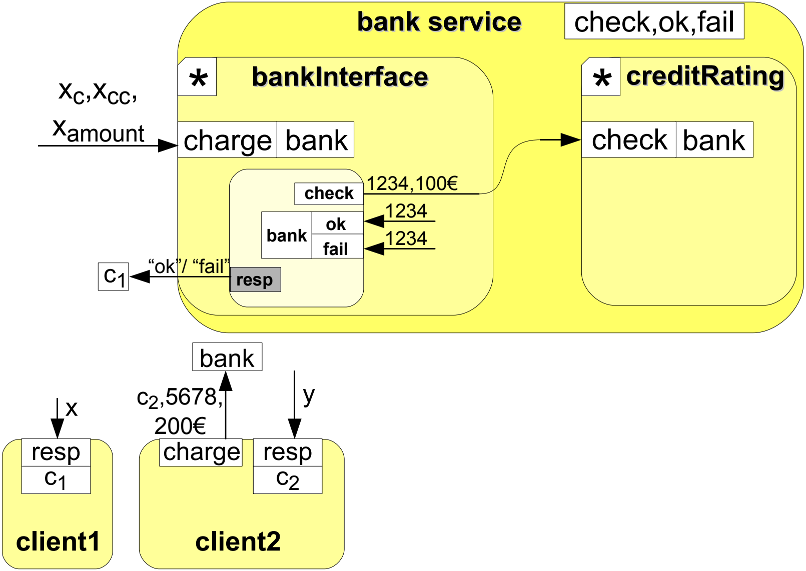
μCOWS^m: *compound* bank service example



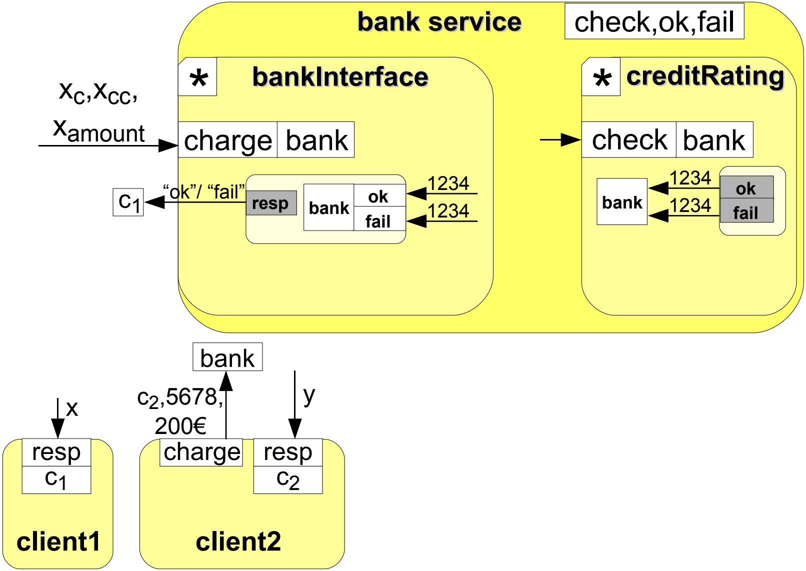
μCOWS^m: *compound* bank service example



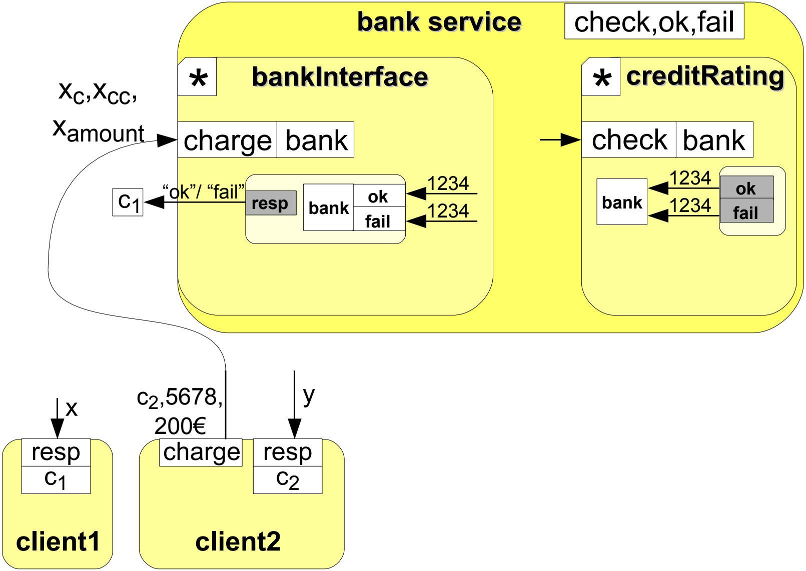
μCOWS^m: *compound* bank service example



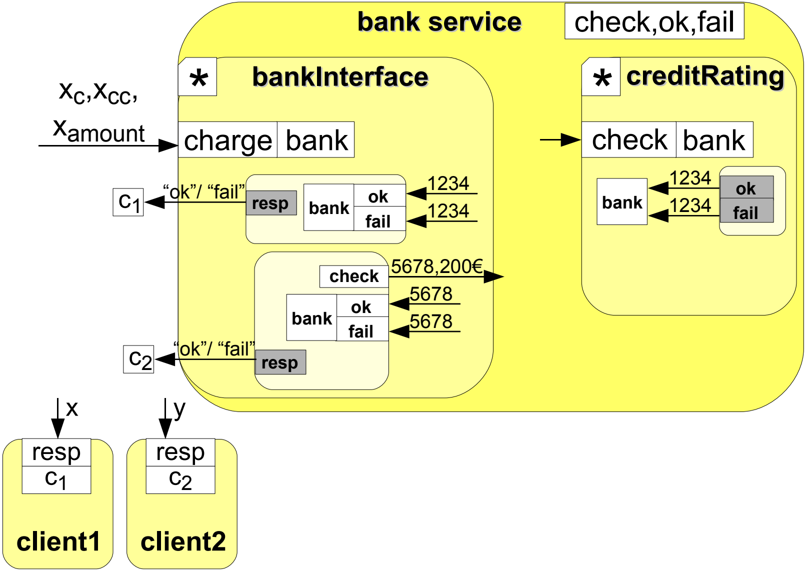
μCOWS^m: *compound* bank service example



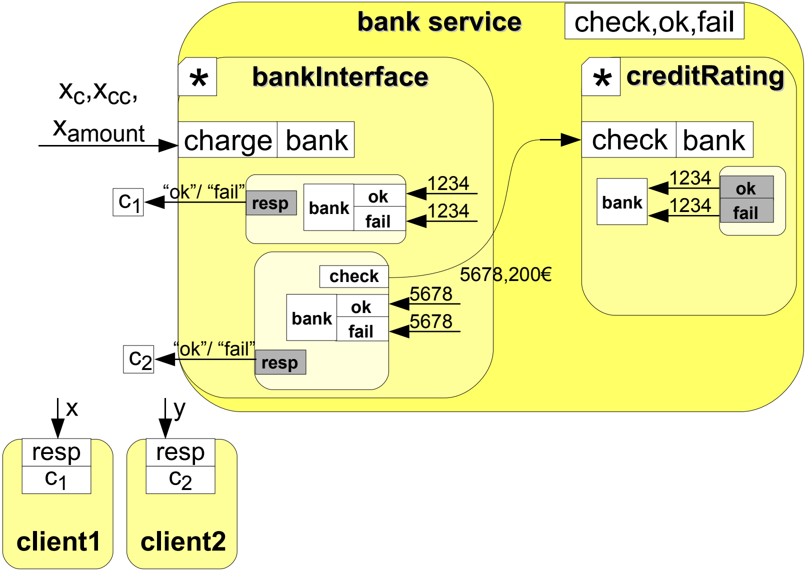
μCOWS^m: *compound* bank service example



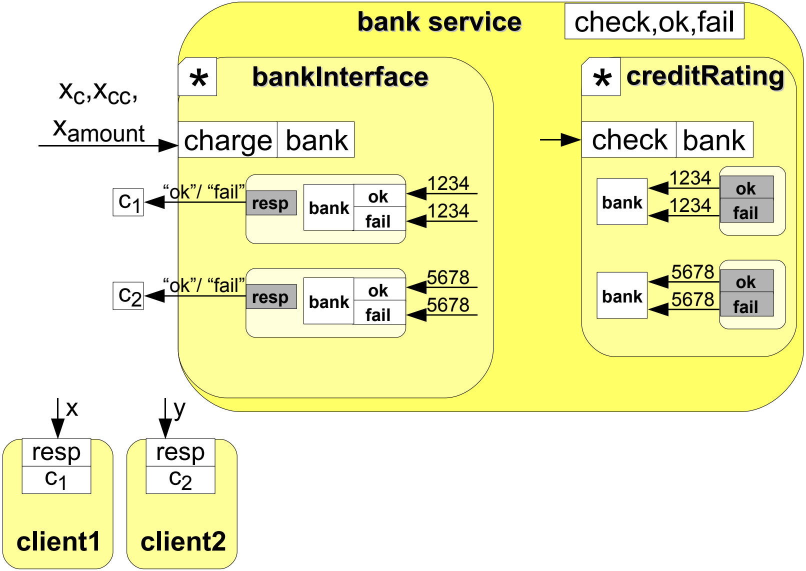
μCOWS^m: *compound* bank service example



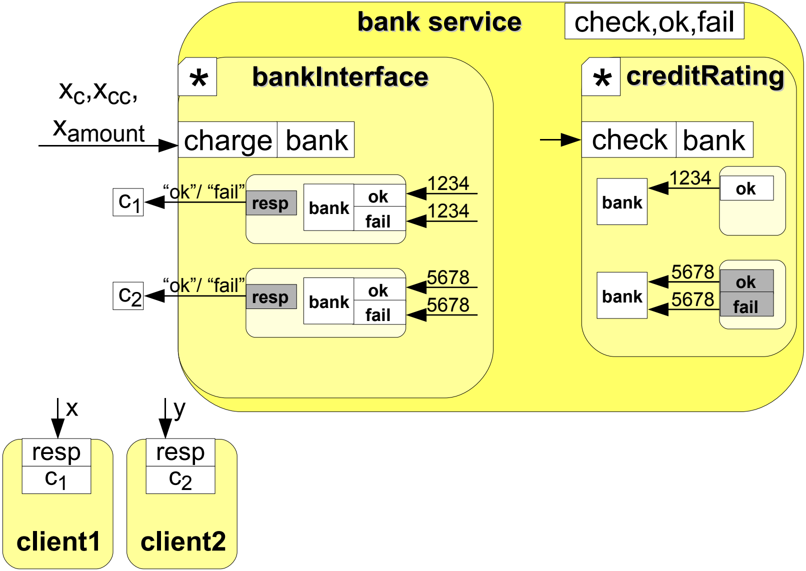
μCOWS^m: *compound* bank service example



μCOWS^m: *compound* bank service example



μCOWS^m: *compound* bank service example



bank service

check,ok,fail

bankInterface

*

$x_c, x_{cc},$

x_{amount}

charge | bank

c_1

"ok"/ "fail"

resp

bank

ok

fail

1234

1234

c_2

"ok"/ "fail"

resp

bank

ok

fail

5678

5678

creditRating

*

check | bank

bank

1234

ok

bank

5678

5678

ok

fail

x

y

resp

c_1

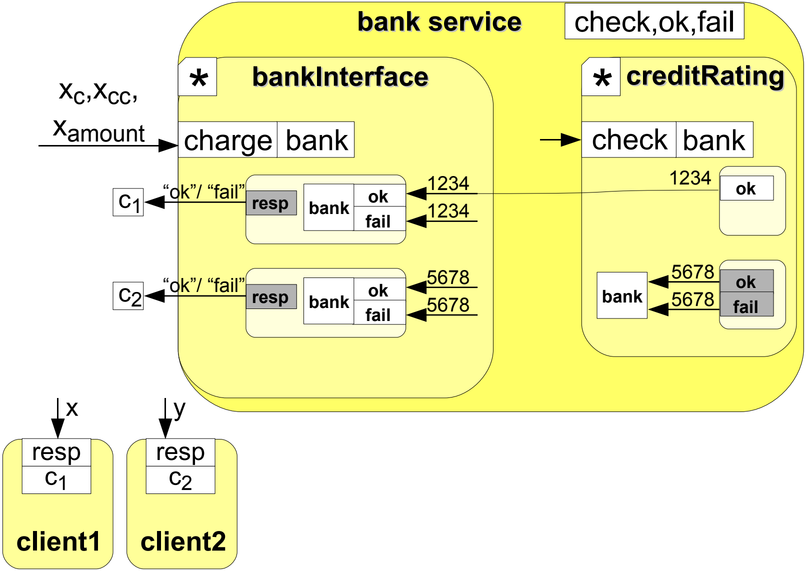
client1

resp

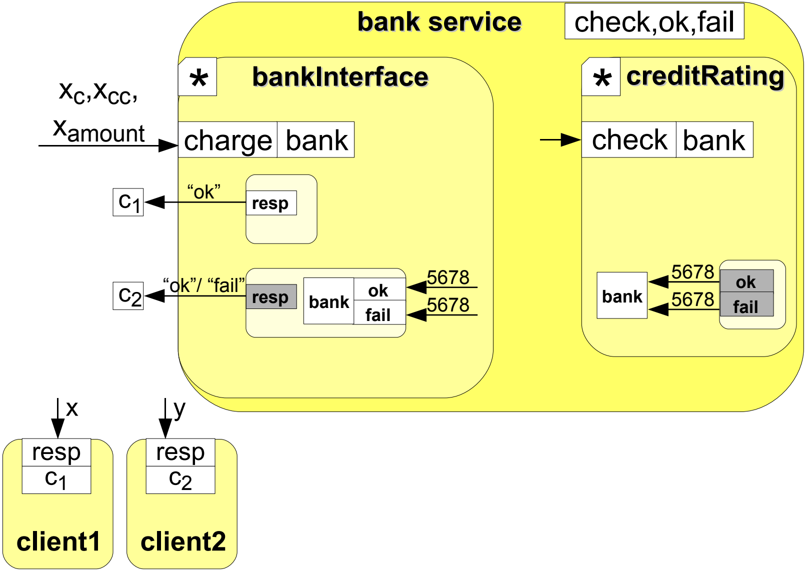
c_2

client2

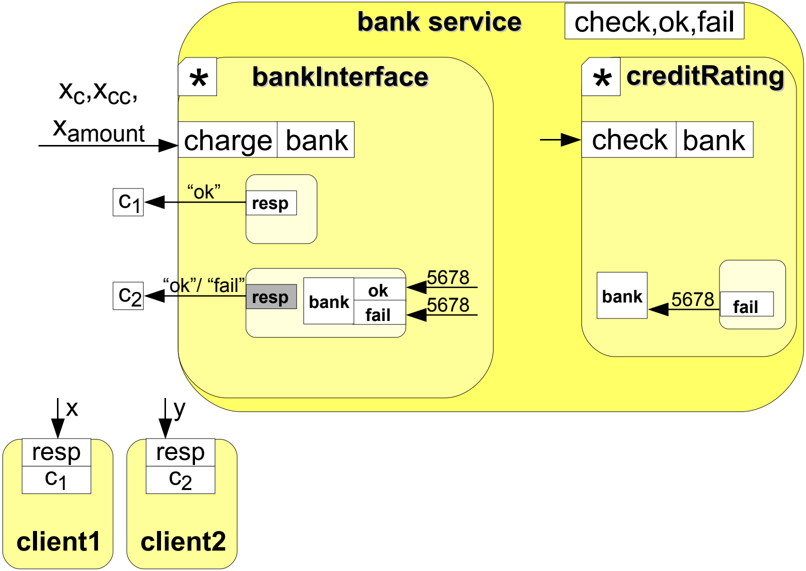
μCOWS^m: *compound* bank service example



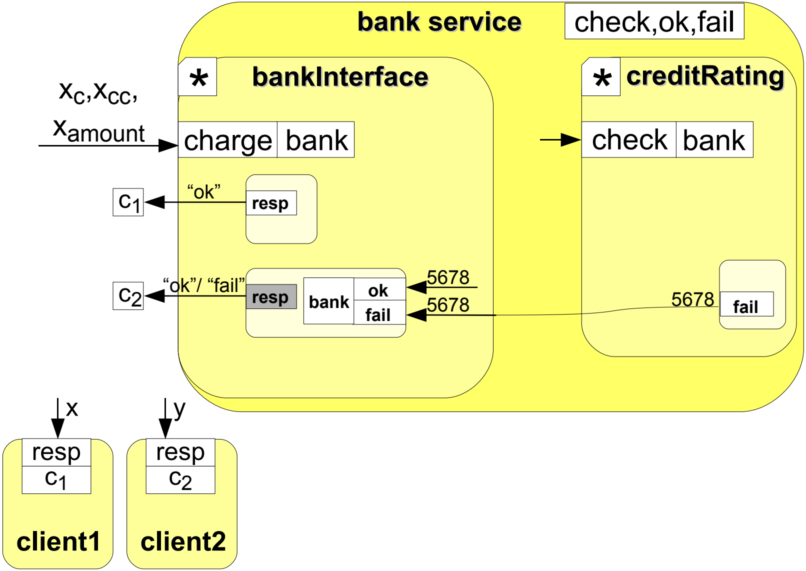
μCOWS^m: *compound* bank service example



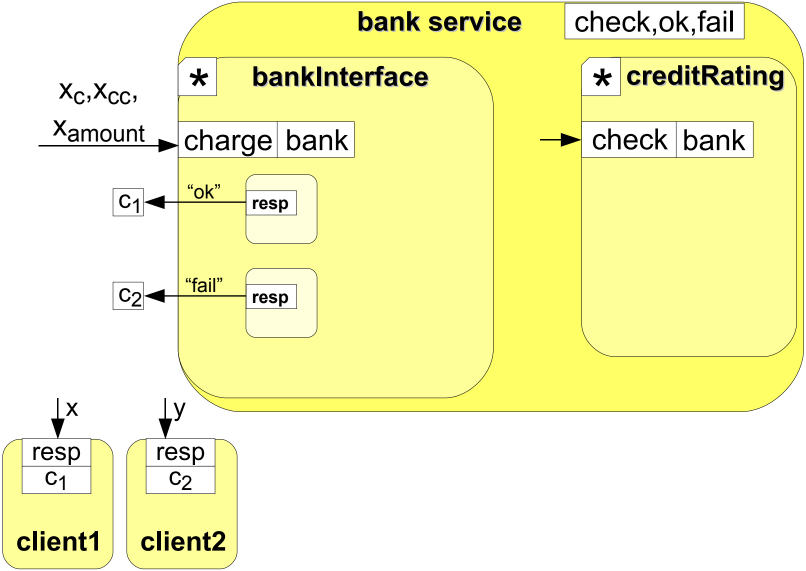
μCOWS^m: *compound* bank service example



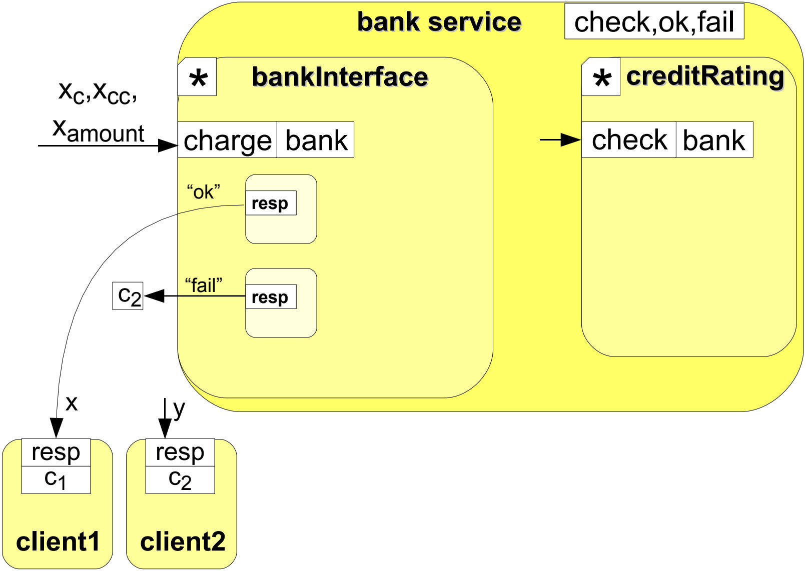
μCOWS^m: *compound* bank service example



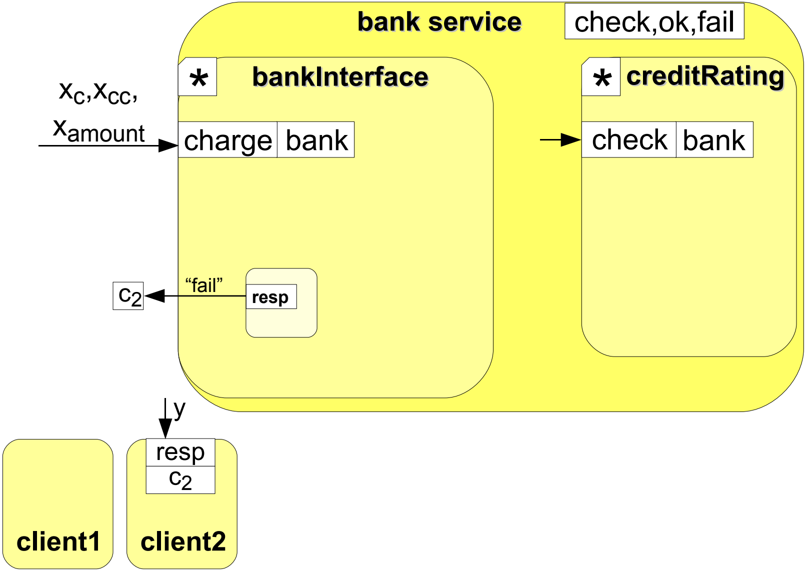
μCOWS^m: *compound* bank service example



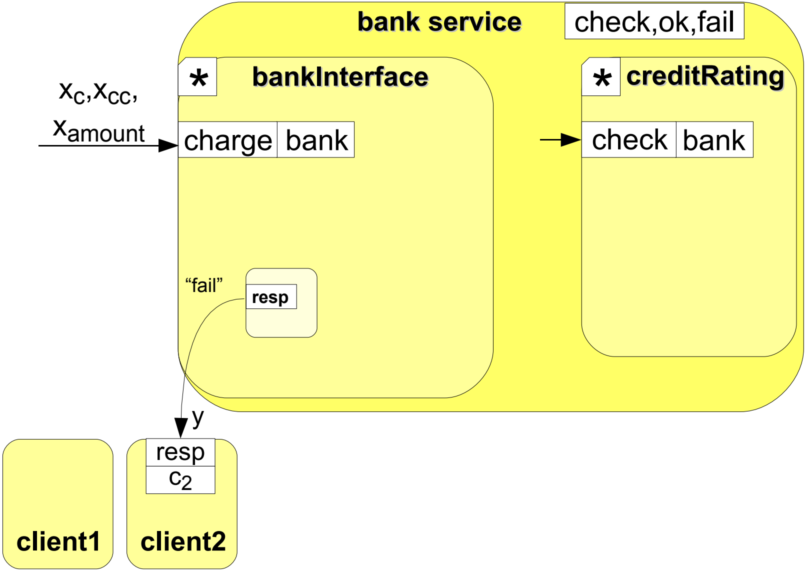
μCOWS^m: *compound* bank service example



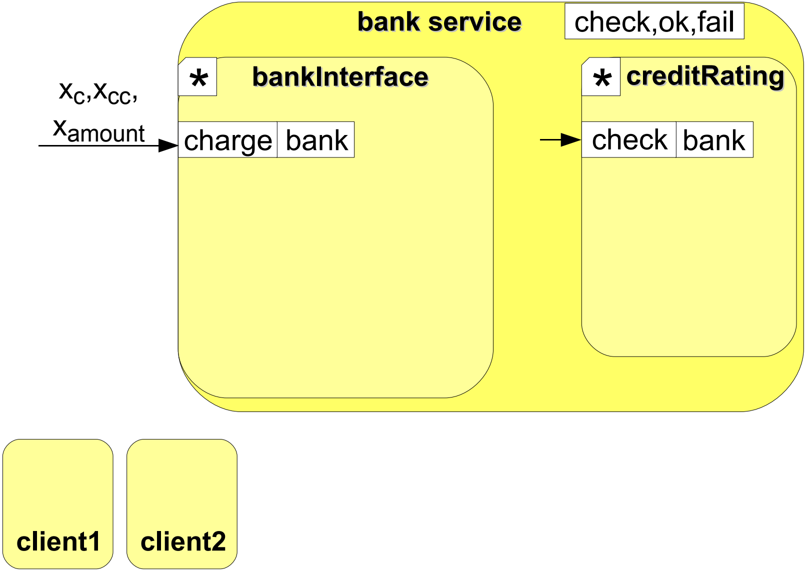
μCOWS^m: *compound* bank service example



μCOWS^m: *compound* bank service example



μCOWS^m: *compound* bank service example



From μCOWS^m to μCOWS

μCOWS^m

From μCOWS^m to μCOWS

μCOWS^m

+

Priority in the parallel composition

From μCOWS^m to μCOWS

μCOWS^m

$+$

Priority in the parallel composition

$=$

μCOWS

μCOWS: why priority in the parallel composition?

1. To deal with *conflicting receives*
   - e.g. in case of multiple start activities
2. Parallel composition with priority can be used (together with pattern-matching) as a *coordination mechanism*
   - e.g. to model default behaviours, transparent session joining, . . .

We use a novel combination of *dynamic* priority with *local* pre-emption

dynamic priority: priority values of activities can change as systems evolve

local pre-emption: priorities have a local scope, i.e. prioritised activities can only pre-empt activities in the same scope

μCOWS: why priority in the parallel composition?

1. To deal with *conflicting receives*
   - e.g. in case of multiple start activities
2. Parallel composition with priority can be used (together with pattern-matching) as a *coordination mechanism*
   - e.g. to model default behaviours, transparent session joining, …

We use a novel combination of *dynamic* priority with *local* pre-emption

dynamic priority: priority values of activities can change as systems evolve

local pre-emption: priorities have a local scope, i.e. prioritised activities can only pre-empt activities in the same scope

μCOWS

Syntax & structural congruence

μCOWS syntax and the set of laws defining its structural congruence coincide with that of μCOWS^m

Labelled transition relation $\xrightarrow{\alpha}$

Label α is now generated by the following grammar:

$$\alpha \quad ::= \quad \mathrm{n} \lhd \bar{v} \quad | \quad \mathrm{n} \rhd \bar{w} \quad | \quad \mathrm{n}\, \sigma\, \ell\, \bar{v}$$

where ℓ is a natural number

μCOWS

Syntax & structural congruence

μCOWS syntax and the set of laws defining its structural congruence coincide with that of μCOWS^m

Labelled transition relation $\xrightarrow{\alpha}$

Label α is now generated by the following grammar:

$$\alpha \quad ::= \quad \mathrm{n} \lhd \bar{v} \quad | \quad \mathrm{n} \rhd \bar{w} \quad | \quad \mathrm{n}\, \sigma\, \ell\, \bar{v}$$

where ℓ is a natural number

μCOWS: Parallel composition with priority

- Communication takes place when two parallel services perform matching receive and invoke activities
- If more then one matching is possible the receive that needs fewer substitutions is selected to progress

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \qquad \mathrm{noConf}(s_1 \mid s_2, \mathrm{n}, \bar{v}, |\sigma|)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\ \sigma\ |\sigma|\ \bar{v}} s_1' \mid s_2'}$$

Conflicting receives predicate

$\mathrm{noConf}(s, \mathrm{n}, \bar{v}, \ell)$ checks existence of potential communication conflicts, i.e. the ability of s of performing a receive activity matching $\bar{v}$ over the endpoint n that generates a substitution with fewer pairs than ℓ

μCOWS: Parallel composition with priority

- Communication takes place when two parallel services perform matching receive and invoke activities
- If more then one matching is possible the receive that needs fewer substitutions is selected to progress

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \qquad \mathrm{noConf}(s_1 \mid s_2, \mathrm{n}, \bar{v}, |\sigma|)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\ \sigma\ |\sigma|\ \bar{v}} s_1' \mid s_2'}$$

Conflicting receives predicate

$\mathrm{noConf}(s, \mathrm{n}, \bar{v}, \ell)$ checks existence of potential communication conflicts, i.e. the ability of s of performing a receive activity matching $\bar{v}$ over the endpoint n that generates a substitution with fewer pairs than ℓ

μCOWS: Parallel composition with priority

- Communication takes place when two parallel services perform matching receive and invoke activities
- If more then one matching is possible the receive that needs fewer substitutions is selected to progress

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \qquad \mathrm{noConf}(s_1 \mid s_2, \mathrm{n}, \bar{v}, |\sigma|)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\ \sigma\ |\sigma|\ \bar{v}} s_1' \mid s_2'}$$

Conflicting receives predicate

$\mathrm{noConf}(s, \mathrm{n}, \bar{v}, \ell)$ checks existence of potential communication conflicts, i.e. the ability of s of performing a receive activity matching $\bar{v}$ over the endpoint n that generates a substitution with fewer pairs than ℓ

μCOWS: Parallel composition with priority

- Communication takes place when two parallel services perform matching receive and invoke activities
- If more then one matching is possible the receive that needs fewer substitutions is selected to progress

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \qquad \mathrm{noConf}(s_1 \mid s_2, \mathrm{n}, \bar{v}, |\sigma|)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\, \sigma\, |\sigma|\, \bar{v}} s_1' \mid s_2'}$$

Conflicting receives predicate (inductive definition, part 1/2)

$$\mathrm{noConf}(\mathbf{kill}(k), \mathrm{n}, \bar{v}, \ell) = \mathrm{noConf}(\mathrm{u}!\bar{\epsilon}, \mathrm{n}, \bar{v}, \ell) = \mathbf{true}$$

$$\mathrm{noConf}(\textstyle\sum_{i=1}^{r} \mathrm{n}_i?\bar{w}_i.s_i, \mathrm{n}, \bar{v}, \ell) = \begin{cases} \mathbf{false} & \text{if } \exists\, i \,.\, \mathrm{n}_i = \mathrm{n} \wedge |\mathcal{M}(\bar{w}_i, \bar{v})| < \ell \\ \mathbf{true} & \text{otherwise} \end{cases}$$

μCOWS: Parallel composition with priority

- Communication takes place when two parallel services perform matching receive and invoke activities
- If more then one matching is possible the receive that needs fewer substitutions is selected to progress

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \qquad \mathrm{noConf}(s_1 \mid s_2, \mathrm{n}, \bar{v}, |\sigma|)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\,\sigma\,|\sigma|\,\bar{v}} s_1' \mid s_2'}$$

Conflicting receives predicate (inductive definition, part 2/2)

$$\mathrm{noConf}(s \mid s', \mathrm{n}, \bar{v}, \ell) = \mathrm{noConf}(s, \mathrm{n}, \bar{v}, \ell) \wedge \mathrm{noConf}(s', \mathrm{n}, \bar{v}, \ell)$$

$$\mathrm{noConf}([u]\, s, \mathrm{n}, \bar{v}, \ell) = \begin{cases} \mathrm{noConf}(s, \mathrm{n}, \bar{v}, \ell) & \text{if } u \notin \mathrm{n} \\ \mathbf{true} & \text{otherwise} \end{cases}$$

$$\mathrm{noConf}(\{|s|\}, \mathrm{n}, \bar{v}, \ell) = \mathrm{noConf}(*\, s, \mathrm{n}, \bar{v}, \ell) = \mathrm{noConf}(s, \mathrm{n}, \bar{v}, \ell)$$

μCOWS: Parallel composition with priority

- Execution of parallel services is interleaved, when no communication is involved:

$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq \mathrm{n}\,\sigma\,\ell\,\bar{v}}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

- In case of communications, the receive activity with greater priority progresses:

$$\frac{s_1 \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s_1' \qquad \mathrm{noConf}(s_2, \mathrm{n}, \bar{v}, \ell)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s_1' \mid s_2}$$

μCOWS: Parallel composition with priority

- Execution of parallel services is interleaved, when no communication is involved:

$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq \mathrm{n}\,\sigma\,\ell\,\bar{v}}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

- In case of communications, the receive activity with greater priority progresses:

$$\frac{s_1 \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s_1' \qquad \mathrm{noConf}(s_2, \mathrm{n}, \bar{v}, \ell)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s_1' \mid s_2}$$

μCOWS: Parallel composition with priority

- Execution of parallel services is interleaved, when no communication is involved:

$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq \mathtt{n}\,\sigma\,\ell\,\bar{v}}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

- In case of communications, the receive activity with greater priority progresses:

$$\frac{s_1 \xrightarrow{\mathtt{n}\,\sigma\,\ell\,\bar{v}} s_1' \qquad \mathrm{noConf}(s_2, \mathtt{n}, \bar{v}, \ell)}{s_1 \mid s_2 \xrightarrow{\mathtt{n}\,\sigma\,\ell\,\bar{v}} s_1' \mid s_2}$$

μCOWS: Delimitation

- Rules for delimitation are tailored to deal with labels $\mathrm{n}\,\sigma\,\ell\,\bar{v}$

$$\frac{s \xrightarrow{\mathrm{n}\,\sigma \uplus \{x \mapsto v\}\,\ell\,\bar{v}} s'}{[x]\,s \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s' \cdot \{x \mapsto v\}} \qquad \frac{s \xrightarrow{\alpha} s' \quad u \notin \mathrm{u}(\alpha)}{[u]\,s \xrightarrow{\alpha} [u]\,s'}$$

where

- $\mathrm{u}(\alpha)$ is extended with $\mathrm{u}(\mathrm{n}\,\sigma\,\ell\,\bar{v}) = \mathrm{u}(\sigma)$

μCOWS: Delimitation

- Rules for delimitation are tailored to deal with labels $\mathrm{n}\,\sigma\,\ell\,\bar{v}$

$$\frac{s \xrightarrow{\mathrm{n}\,\sigma \uplus \{x \mapsto v\}\,\ell\,\bar{v}} s'}{[x]\,s \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s' \cdot \{x \mapsto v\}} \qquad \frac{s \xrightarrow{\alpha} s' \quad u \notin \mathrm{u}(\alpha)}{[u]\,s \xrightarrow{\alpha} [u]\,s'}$$

where

- $\mathrm{u}(\alpha)$ is extended with $\mathrm{u}(\mathrm{n}\,\sigma\,\ell\,\bar{v}) = \mathrm{u}(\sigma)$

μCOWS operational semantics

Labelled transition rules

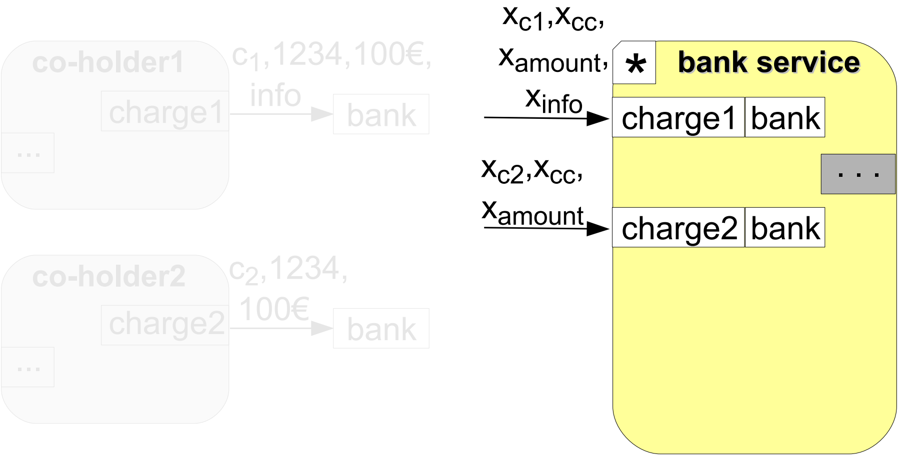
$$\frac{[\![\bar{\epsilon}]\!] = \bar{v}}{\mathrm{n}!\bar{\epsilon} \xrightarrow{\mathrm{n} \lhd \bar{v}} \mathbf{0}} \qquad \frac{1 \le j \le r}{\sum_{i=1}^{r} \mathrm{n}_i?\bar{w}_i.s_i \xrightarrow{\mathrm{n}_j \rhd \bar{w}_j} s_j} \qquad \frac{s \xrightarrow{\alpha} s' \quad u \notin \mathrm{u}(\alpha)}{[u]\, s \xrightarrow{\alpha} [u]\, s'} \qquad \frac{s \equiv \xrightarrow{\alpha} \equiv s'}{s \xrightarrow{\alpha} s'}$$

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \quad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \quad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \quad \mathrm{noConf}(s_1 \mid s_2, \mathrm{n}, \bar{v}, |\sigma|)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\, \sigma\, |\sigma|\, \bar{v}} s_1' \mid s_2'}$$

$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq \mathrm{n}\, \sigma\, \ell\, \bar{v}}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2} \qquad \frac{s_1 \xrightarrow{\mathrm{n}\, \sigma\, \ell\, \bar{v}} s_1' \qquad \mathrm{noConf}(s_2, \mathrm{n}, \bar{v}, \ell)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\, \sigma\, \ell\, \bar{v}} s_1' \mid s_2}$$

$$\frac{s \xrightarrow{\mathrm{n}\, \sigma \uplus \{x \mapsto v\}\, \ell\, \bar{v}} s'}{[x]\, s \xrightarrow{\mathrm{n}\, \sigma\, \ell\, \bar{v}} s' \cdot \{x \mapsto v\}}$$

μCOWS: *joint account* service example

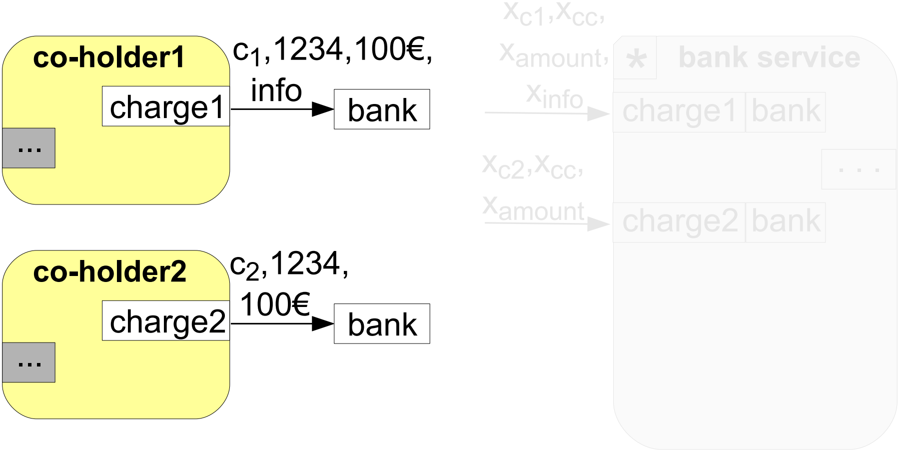


$$* \,[x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}]\,(\,\mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info}\rangle . s_1 \;\mid\; \mathrm{bank} \bullet \mathrm{charge2}?\langle x_{c2}, x_{cc}, x_{amount}\rangle . s_2\,)$$

$$\mid\; (\,\mathrm{bank} \bullet \mathrm{charge1}!\langle c_1, 1234, 100\text{€}, \mathrm{info}\rangle \;\mid\; s_1'\,)$$

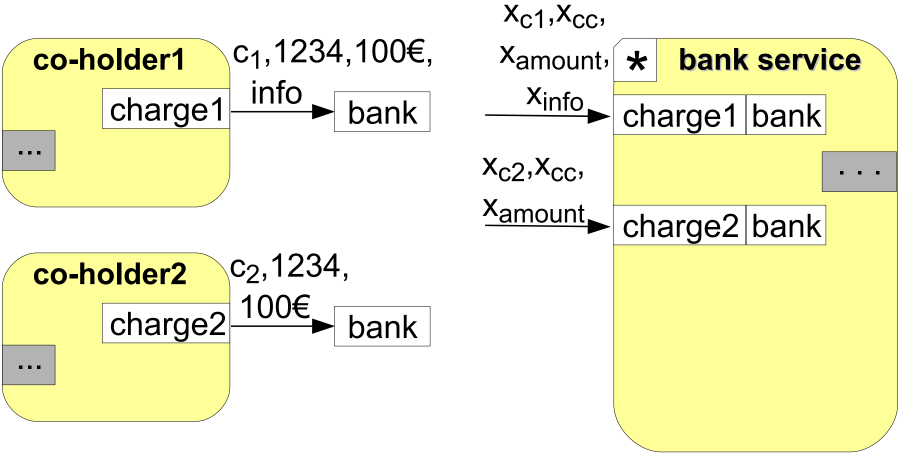
$$\mid\; (\,\mathrm{bank} \bullet \mathrm{charge2}!\langle c_2, 1234, 100\text{€}\rangle \;\mid\; s_2'\,)$$

μCOWS: *joint account* service example



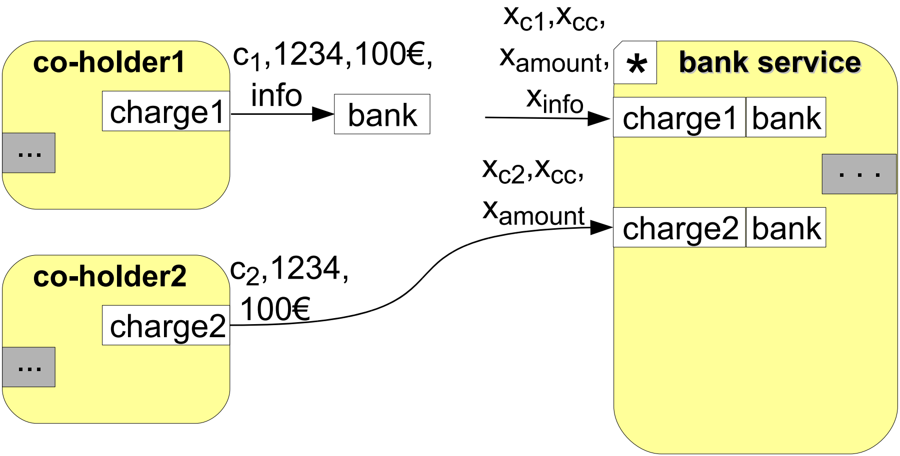
$$
\begin{aligned}
& * \; [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (\mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
& \qquad \mid \mathrm{bank} \bullet \mathrm{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
\mid \; & (\mathrm{bank} \bullet \mathrm{charge1}!\langle c_1, 1234, 100\text{€}, \mathrm{info} \rangle \mid s'_1 \,) \\
\mid \; & (\mathrm{bank} \bullet \mathrm{charge2}!\langle c_2, 1234, 100\text{€} \rangle \mid s'_2 \,)
\end{aligned}
$$

μCOWS: *joint account* service example



$$
\begin{array}{l}
*\,[x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}]\,(\,\mathrm{bank}\bullet\mathrm{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info}\rangle.s_1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \mid\ \mathrm{bank}\bullet\mathrm{charge2}?\langle x_{c2}, x_{cc}, x_{amount}\rangle.s_2\,) \\
\mid\ (\,\mathrm{bank}\bullet\mathrm{charge1}!\langle c_1, 1234, 100€, \mathrm{info}\rangle \mid s_1'\,) \\
\mid\ (\,\mathrm{bank}\bullet\mathrm{charge2}!\langle c_2, 1234, 100€\rangle \mid s_2'\,)
\end{array}
$$

μCOWS: *joint account* service example



co-holder1

charge1

...

c_1,1234,100€, info

bank

x_{c1},x_{cc}, x_{amount}, x_{info}

*

bank service

charge1 bank

. . .

x_{c2},x_{cc}, x_{amount}

charge2 bank

co-holder2

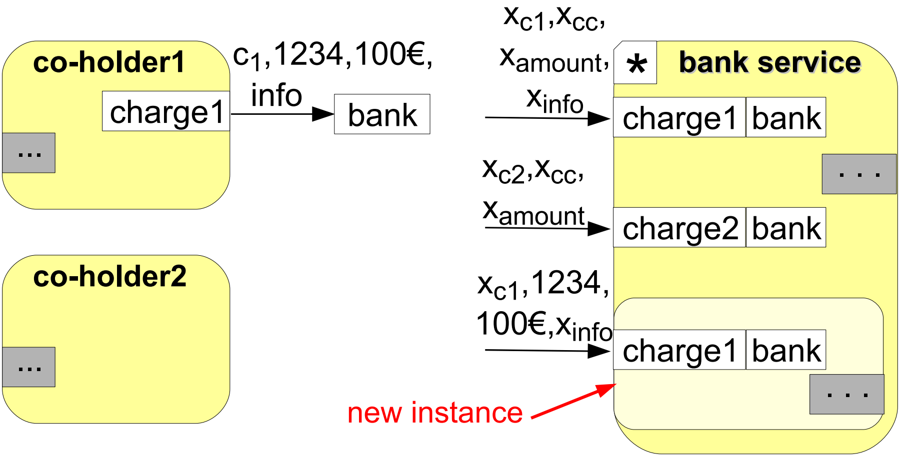
charge2

...

c_2,1234, 100€

$$\begin{aligned}
&* \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (\, \text{bank} \bullet \text{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
&\qquad\qquad\qquad\qquad\qquad \mid \; \text{bank} \bullet \text{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
&\mid \; (\, \text{bank} \bullet \text{charge1}!\langle c_1, 1234, 100\text{€}, \text{info} \rangle \;\mid\; s'_1 \,) \\
&\mid \; (\, \text{bank} \bullet \text{charge2}!\langle c_2, 1234, 100\text{€} \rangle \;\mid\; s'_2 \,)
\end{aligned}$$

μCOWS: *joint account* service example



co-holder1

charge1

c_1,1234,100€, info

bank

...

co-holder2

...

x_{c1},x_{cc}, x_{amount}, x_{info}

$*$

bank service

charge1 bank

. . .

x_{c2},x_{cc}, x_{amount}

charge2 bank

x_{c1},1234, 100€,x_{info}

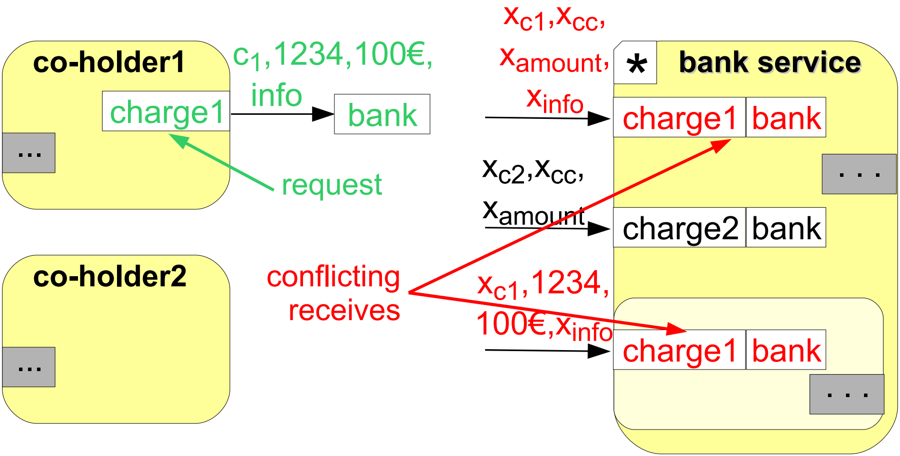
charge1 bank

. . .

new instance

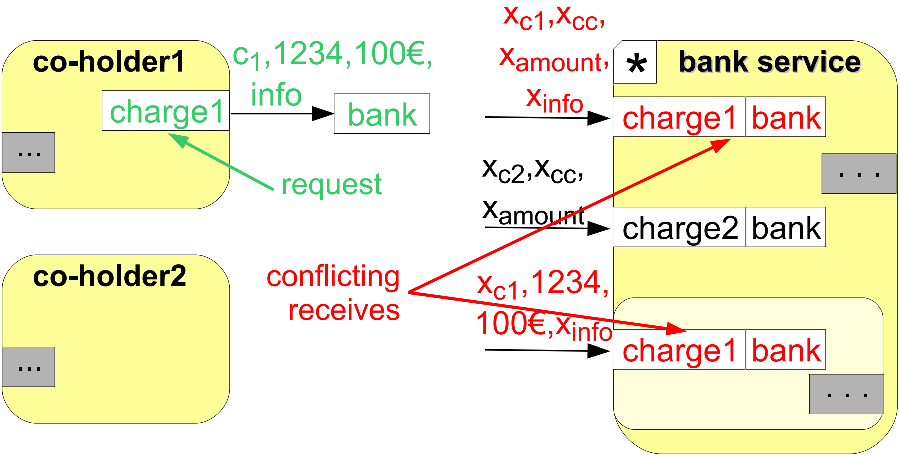
$$\begin{aligned}
&* \,[x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}]\, (\,\mathsf{bank} \bullet \mathsf{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info}\rangle . s_1 \\
&\qquad\qquad\qquad\qquad \mid \mathsf{bank} \bullet \mathsf{charge2}?\langle x_{c2}, x_{cc}, x_{amount}\rangle . s_2\,) \\
&\mid (\,\mathsf{bank} \bullet \mathsf{charge1}?\langle x_{c1}, 1234, 100\text{€}, x_{info}\rangle . s_1 \mid s_2\,) \cdot \{\cdots \mapsto \cdots\} \\
&\mid (\,\mathsf{bank} \bullet \mathsf{charge1}!\langle c_1, 1234, 100\text{€}, \mathsf{info}\rangle \mid s'_1\,) \mid (\,s'_2\,)
\end{aligned}$$

μCOWS: *joint account* service example



$$
\begin{aligned}
&* \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (\mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
&\qquad\qquad\qquad\qquad \mid \mathrm{bank} \bullet \mathrm{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
&\mid \, (\mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, 1234, 100\text{€}, x_{info} \rangle . s_1 \mid s_2 \,) \cdot \{ \cdots \mapsto \cdots \} \\
&\mid \, (\mathrm{bank} \bullet \mathrm{charge1}!\langle c_1, 1234, 100\text{€}, \mathrm{info} \rangle \mid s_1' \,) \mid (\, s_2' \,)
\end{aligned}
$$

μCOWS: *joint account* service example

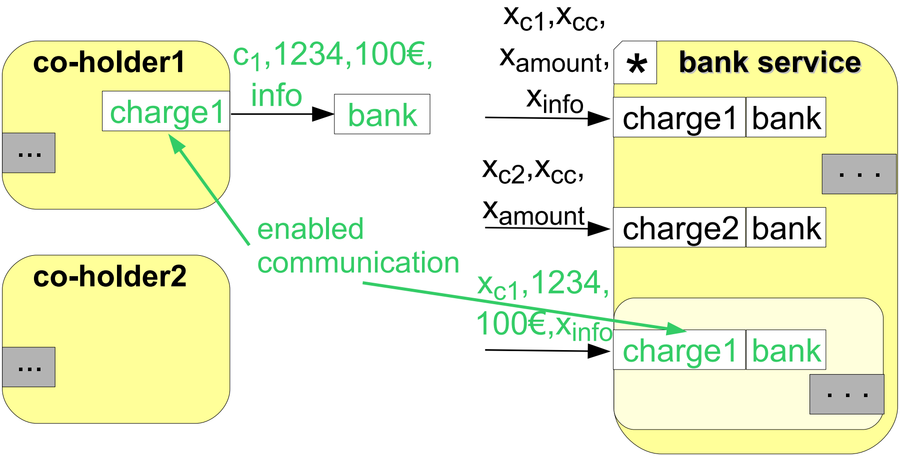


Multiple start activities

The service can receive multiple messages in a statically unpredictable order s.t.

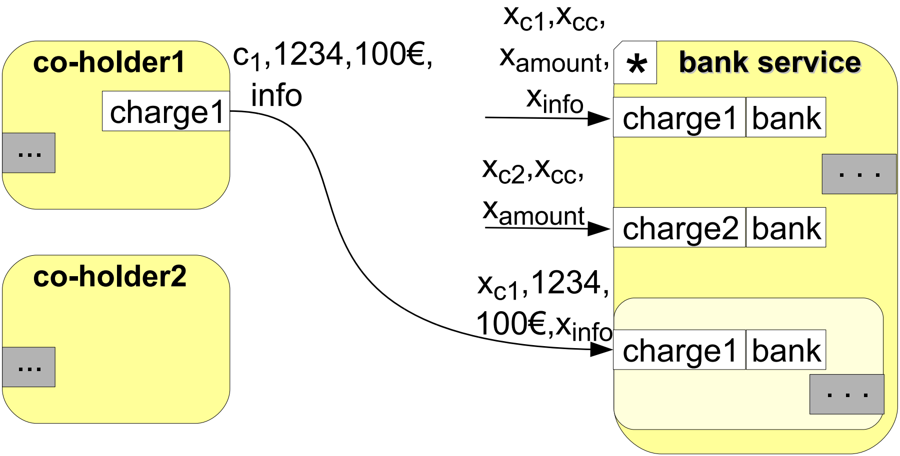
- the first incoming message triggers creation of a service instance
- subsequent messages are delivered to the created instance

μCOWS: *joint account* service example



$$
\begin{aligned}
&* \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (\, \mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
&\qquad\qquad\qquad\qquad \mid \; \mathrm{bank} \bullet \mathrm{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
&\mid \; (\, \mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, 1234, 100\text{€}, x_{info} \rangle . s_1 \; \mid \; s_2 \,) \cdot \{ \cdots \mapsto \cdots \} \\
&\mid \; (\, \mathrm{bank} \bullet \mathrm{charge1}!\langle c_1, 1234, 100\text{€}, \mathrm{info} \rangle \; \mid \; s_1' \,) \; \mid \; (\, s_2' \,)
\end{aligned}
$$

μCOWS: *joint account* service example



co-holder1

charge1

...

c_1,1234,100€, info

co-holder2

...

x_{c1},x_{cc}, x_{amount}, x_{info}

*****

bank service

charge1 bank

. . .

x_{c2},x_{cc}, x_{amount}

charge2 bank

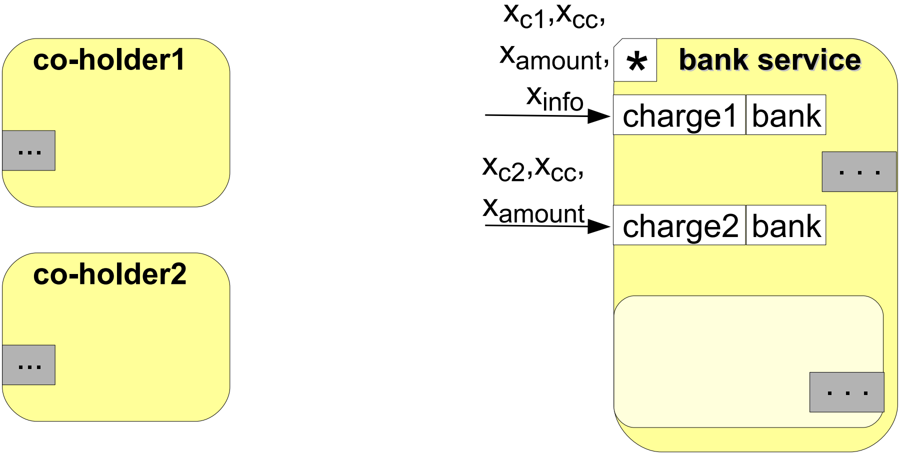
x_{c1},1234, 100€,x_{info}

charge1 bank

. . .

$$
\begin{aligned}
&* \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (\, \mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
&\qquad\qquad\qquad\qquad\qquad \mid \mathrm{bank} \bullet \mathrm{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
&\mid (\, \mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, 1234, 100\text{€}, x_{info} \rangle . s_1 \mid s_2 \,) \cdot \{ \cdots \mapsto \cdots \} \\
&\mid (\, \mathrm{bank} \bullet \mathrm{charge1}!\langle c_1, 1234, 100\text{€}, \mathrm{info} \rangle \mid s'_1 \,) \mid (\, s'_2 \,)
\end{aligned}
$$

μCOWS: *joint account* service example



$$
\begin{aligned}
* \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (& \text{bank} \bullet \text{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
& \mid \text{bank} \bullet \text{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
\mid \; (s_1 \mid s_2) \cdot \{ \cdots \mapsto \cdots \} & \\
\mid \; (s_1') \mid (s_2') &
\end{aligned}
$$

Parallel with priority as a coordination mechanism

Default behaviour

Consider a service providing mathematical functionalities
e.g. sum of two integers between 0 and 5

$$
\begin{aligned}
* [x, y, z] \, (\, & \mathit{math} \bullet \mathit{sum}?\langle x, y, z \rangle . \, x \bullet \mathit{resp}!\langle \mathit{error} \rangle \\
& + \; \mathit{math} \bullet \mathit{sum}?\langle x, 0, 0 \rangle . \, x \bullet \mathit{resp}!\langle 0 \rangle \\
& + \; \mathit{math} \bullet \mathit{sum}?\langle x, 0, 1 \rangle . \, x \bullet \mathit{resp}!\langle 1 \rangle \\
& + \; \ldots \; + \; \mathit{math} \bullet \mathit{sum}?\langle x, 5, 5 \rangle . \, x \bullet \mathit{resp}!\langle 10 \rangle \;)
\end{aligned}
$$

In case the two values are not admissible, i.e. they are not integers between 0 and 5, the service replies with the string *error*

Parallel with priority as a coordination mechanism

'Only the first time' behaviour

Consider a service that has a certain behaviour at the first correct invocation and a different behaviour at any incorrect or further invocation (useful, e.g., for compensation handling à la WS-BPEL)

$$\begin{array}{l} p \bullet comp?\langle scopeName\rangle.\,\langle\text{compensation of } scopeName\rangle \\ \mid\ *\,[x]\, p \bullet comp?\langle x\rangle.\,\langle\text{do nothing}\rangle \end{array}$$

Parallel with priority as a coordination mechanism

‘Blind date’ session joining

Consider a service capable of arranging matches of 4-players online games

$$
\begin{array}{rcl}
\mathit{masterServ} & \triangleq & *\,[x_{game}, x_{player1}, x_{player2}, x_{player3}, x_{player4}] \\
& & \quad \mathit{master} \bullet \mathit{join}?\langle x_{game}, x_{player1}\rangle. \\
& & \quad \mathit{master} \bullet \mathit{join}?\langle x_{game}, x_{player2}\rangle. \\
& & \quad \mathit{master} \bullet \mathit{join}?\langle x_{game}, x_{player3}\rangle. \\
& & \quad \mathit{master} \bullet \mathit{join}?\langle x_{game}, x_{player4}\rangle. \\
& & \qquad [\mathit{matchId}]\,(\, x_{player1} \bullet \mathit{start}!\langle \mathit{matchId}\rangle \\
& & \qquad\qquad\qquad \mid x_{player2} \bullet \mathit{start}!\langle \mathit{matchId}\rangle \\
& & \qquad\qquad\qquad \mid x_{player3} \bullet \mathit{start}!\langle \mathit{matchId}\rangle \\
& & \qquad\qquad\qquad \mid x_{player4} \bullet \mathit{start}!\langle \mathit{matchId}\rangle\,)
\end{array}
$$

$$\mathit{Player}_i \triangleq \mathit{master} \bullet \mathit{join}!\langle \mathit{poker}, p_i\rangle \mid [x_{id}]\, p_i \bullet \mathit{start}?\langle x_{id}\rangle. \langle \text{rest of } \mathit{Player}_i\rangle$$

$$\mathit{Player}_j \triangleq \mathit{master} \bullet \mathit{join}!\langle \mathit{bridge}, p_j\rangle \mid [x_{id}]\, p_j \bullet \mathit{start}?\langle x_{id}\rangle. \langle \text{rest of } \mathit{Player}_j\rangle$$

It could be hard to render this behaviour with other process calculi

Parallel with priority as a coordination mechanism

‘Blind date’ session joining

Consider a service capable of arranging matches of 4-players online games

$$
\begin{aligned}
\mathit{masterServ} \triangleq\ & *\,[x_{game}, x_{player1}, x_{player2}, x_{player3}, x_{player4}] \\
& \quad \mathit{master} \bullet \mathit{join}?\langle x_{game}, x_{player1}\rangle. \\
& \quad \mathit{master} \bullet \mathit{join}?\langle x_{game}, x_{player2}\rangle. \\
& \quad \mathit{master} \bullet \mathit{join}?\langle x_{game}, x_{player3}\rangle. \\
& \quad \mathit{master} \bullet \mathit{join}?\langle x_{game}, x_{player4}\rangle. \\
& \qquad [\mathit{matchId}]\,(\, x_{player1} \bullet \mathit{start}!\langle \mathit{matchId}\rangle \\
& \qquad\qquad\quad \mid x_{player2} \bullet \mathit{start}!\langle \mathit{matchId}\rangle \\
& \qquad\qquad\quad \mid x_{player3} \bullet \mathit{start}!\langle \mathit{matchId}\rangle \\
& \qquad\qquad\quad \mid x_{player4} \bullet \mathit{start}!\langle \mathit{matchId}\rangle\,)
\end{aligned}
$$

$$\mathit{Player}_i \triangleq \mathit{master} \bullet \mathit{join}!\langle \mathit{poker}, p_i\rangle \mid [x_{id}]\, p_i \bullet \mathit{start}?\langle x_{id}\rangle.\, \langle \text{rest of } \mathit{Player}_i\rangle$$

$$\mathit{Player}_j \triangleq \mathit{master} \bullet \mathit{join}!\langle \mathit{bridge}, p_j\rangle \mid [x_{id}]\, p_j \bullet \mathit{start}?\langle x_{id}\rangle.\, \langle \text{rest of } \mathit{Player}_j\rangle$$

It could be hard to render this behaviour with other process calculi

From μCOWS to COWS

μCOWS

From μCOWS to COWS

μCOWS

+

Termination activities

From μCOWS to COWS

μCOWS

+

Termination activities

=

COWS

COWS: why termination activities?

1. To handle *faults* and enable *compensation*
2. Termination activities can be used as *orchestration mechanisms*
    - E.g. to model the asymmetric parallel composition of Orc (i.e. the *pruning* construct, that prunes threads selectively)

Syntax of COWS

$$\begin{array}{lll}
s ::= & & \text{(services)} \\
 & \mathbf{kill}(k) & \text{(kill)} \\
| & u \bullet u'!\bar{\epsilon} & \text{(invoke)} \\
| & \sum_{i=0}^{r} g_i . s_i & \text{(receive-guarded choice)} \\
| & s \mid s & \text{(parallel composition)} \\
| & \{\!|\, s \,|\!\} & \text{(protection)} \\
| & [e]\, s & \text{(delimitation)} \\
| & *\, s & \text{(replication)} \\
 & & \\
g ::= & & \text{(guards)} \\
 & p \bullet o?\bar{w} & \text{(receive)}
\end{array}$$

(notations)
k: *(killer) labels*
ϵ: *expressions*
x: *variables*
v: *values*
n, p, o: *names*
u: *variables* | *names*
w: *variables* | *values*
e: *labels* | *variables* | *names*

- Killer labels cannot occur within expressions
  $\Rightarrow$ they are not (communicable) values
- Only one binding construct: $[e]\, s$ binds e in the scope s
  - free/bound *elements* (i.e. names/variables/labels) defined accordingly

COWS operational semantics

Additional structural congruence laws

- $\{\!|\mathbf{0}|\!\} \;\equiv\; \mathbf{0} \qquad \{\!|\,\{\!|s|\!\}\,|\!\} \;\equiv\; \{\!|s|\!\} \qquad \{\!|[e]\,s|\!\} \;\equiv\; [e]\,\{\!|s|\!\}$
- $s_1 \mid [e]\,s_2 \;\equiv\; [e]\,(s_1 \mid s_2) \quad \mathrm{if}\ e \notin \mathrm{fe}(s_1) \cup \mathrm{fk}(s_2)$
  - $\mathrm{fe}(s)$ denotes the set of elements occurring free in s
  - $\mathrm{fk}(s)$ denotes the set of free killer labels in s
  - thus, differently from names/variables, the scope of killer labels cannot be extended

Labelled transition relation $\xrightarrow{\alpha}$

Label α is now generated by the following grammar:

$$\alpha \;::=\; \mathrm{n} \lhd \bar{v} \;\mid\; \mathrm{n} \rhd \bar{w} \;\mid\; \mathrm{n}\,\sigma\,\ell\,\bar{v} \;\mid\; k \;\mid\; \dagger$$

COWS: Kill activity

- Activity **kill**(k) forces termination of all unprotected parallel activities inside an enclosing $[k]$, that stops the killing effect

$$\mathbf{kill}(k) \xrightarrow{k} \mathbf{0} \qquad \frac{s_1 \xrightarrow{k} s_1'}{s_1 \mid s_2 \xrightarrow{k} s_1' \mid \mathrm{halt}(s_2)} \qquad \frac{s \xrightarrow{k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'}$$

COWS: Kill activity

- Activity **kill**(k) forces termination of all unprotected parallel activities inside an enclosing $[k]$, that stops the killing effect

$$\mathbf{kill}(k) \xrightarrow{k} \mathbf{0} \qquad \frac{s_1 \xrightarrow{k} s_1'}{s_1 \mid s_2 \xrightarrow{k} s_1' \mid \mathrm{halt}(s_2)} \qquad \frac{s \xrightarrow{k} s'}{[k]\,s \xrightarrow{\dagger} [k]\,s'}$$

Function $\mathrm{halt}(s)$

returns the service obtained by only retaining the protected activities inside s

$$\mathrm{halt}(\mathbf{kill}(k)) \;=\; \mathrm{halt}(\mathtt{u}!\bar{\epsilon}) \;=\; \mathrm{halt}(\,\textstyle\sum_{i=0}^{r} \mathtt{n}_i?\bar{w}_i.s_i) \;=\; \mathbf{0}$$

$$\mathrm{halt}(s_1 \mid s_2) = \mathrm{halt}(s_1) \mid \mathrm{halt}(s_2) \qquad \mathrm{halt}(\{\!|s|\!\}) = \{\!|s|\!\}$$

$$\mathrm{halt}([e]\,s) = [e]\,\mathrm{halt}(s) \qquad \mathrm{halt}(*\,s) = *\,\mathrm{halt}(s)$$

COWS: Kill activity

- Activity **kill**(k) forces termination of all unprotected parallel activities inside an enclosing $[k]$, that stops the killing effect

$$\mathbf{kill}(k) \xrightarrow{k} \mathbf{0} \qquad \frac{s_1 \xrightarrow{k} s_1'}{s_1 \mid s_2 \xrightarrow{k} s_1' \mid \mathrm{halt}(s_2)} \qquad \frac{s \xrightarrow{k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'}$$

- Kill activities are executed *eagerly*

$$\frac{s \xrightarrow{k} s' \quad k \neq e}{[e]\, s \xrightarrow{k} [e]\, s'} \qquad \frac{s \xrightarrow{\dagger} s'}{[e]\, s \xrightarrow{\dagger} [e]\, s'}$$

$$\frac{s \xrightarrow{\alpha} s' \quad e \notin \mathrm{e}(\alpha) \quad \alpha \neq k, \dagger \quad \mathrm{noKill}(s, e)}{[e]\, s \xrightarrow{\alpha} [e]\, s'}$$

COWS: Kill activity

- Activity **kill**(k) forces termination of all unprotected parallel activities inside an enclosing $[k]$, that stops the killing effect
- Kill activities are executed *eagerly*

$$\frac{s \xrightarrow{k} s' \quad k \neq e}{[e]\, s \xrightarrow{k} [e]\, s'} \qquad\qquad \frac{s \xrightarrow{\dagger} s'}{[e]\, s \xrightarrow{\dagger} [e]\, s'}$$

$$\frac{s \xrightarrow{\alpha} s' \quad e \notin \mathrm{e}(\alpha) \quad \alpha \neq k, \dagger \quad \mathrm{noKill}(s, e)}{[e]\, s \xrightarrow{\alpha} [e]\, s'}$$

Predicate $\mathrm{noKill}(s, e)$ (part 1/2)

checks the ability of s of immediately performing a kill activity

$$\mathrm{noKill}(s, e) \;=\; \textbf{true} \quad \text{if } \mathrm{fk}(e) = \emptyset \qquad \mathrm{noKill}(\textbf{kill}(k'), k) \;=\; \textbf{true} \quad \text{if } k \neq k'$$

$$\mathrm{noKill}(\textbf{kill}(k), k) \;=\; \textbf{false} \qquad \mathrm{noKill}(\mathrm{u}!\bar{\epsilon}, k) \;=\; \mathrm{noKill}(\textstyle\sum_{i=0}^{r} \mathrm{n}_i?\bar{w}_i.s_i, k) \;=\; \textbf{true}$$

COWS: Kill activity

- Activity **kill**(k) forces termination of all unprotected parallel activities inside an enclosing $[k]$, that stops the killing effect
- Kill activities are executed *eagerly*

$$\frac{s \xrightarrow{k} s' \quad k \neq e}{[e]\, s \xrightarrow{k} [e]\, s'} \qquad \frac{s \xrightarrow{\dagger} s'}{[e]\, s \xrightarrow{\dagger} [e]\, s'}$$

$$\frac{s \xrightarrow{\alpha} s' \quad e \notin \mathrm{e}(\alpha) \quad \alpha \neq k, \dagger \quad \mathrm{noKill}(s, e)}{[e]\, s \xrightarrow{\alpha} [e]\, s'}$$

Predicate $\mathrm{noKill}(s, e)$ (part 2/2)

checks the ability of s of immediately performing a kill activity

$\mathrm{noKill}(s \mid s', k) = \mathrm{noKill}(s, k) \wedge \mathrm{noKill}(s', k)$ $\quad$ $\mathrm{noKill}([e]\, s, k) = \mathrm{noKill}(s, k)$ if $e \neq k$

$\mathrm{noKill}([k]\, s, k) = \textbf{true}$ $\quad$ $\mathrm{noKill}(\{\!|s|\!\}, k) = \mathrm{noKill}(* \, s, k) = \mathrm{noKill}(s, k)$

COWS: Kill activity

- Activity **kill**(k) forces termination of all unprotected parallel activities inside an enclosing $[k]$, that stops the killing effect
- Kill activities are executed *eagerly*
- $\{\!|\cdot|\!\}$ protects activities from the effect of a forced termination

$$\frac{s \xrightarrow{\alpha} s'}{\{\!|s|\!\} \xrightarrow{\alpha} \{\!|s'|\!\}}$$

COWS operational semantics: labelled transition rules

$$\frac{[\![\bar{\epsilon}]\!] = \bar{v}}{\mathrm{n}!\bar{\epsilon} \xrightarrow{\mathrm{n} \lhd \bar{v}} \mathbf{0}} \qquad \frac{1 \le j \le r}{\sum_{i=1}^{r} \mathrm{n}_i?\bar{w}_i.s_i \xrightarrow{\mathrm{n}_j \rhd \bar{w}_j} s_j} \qquad \frac{s \equiv \xrightarrow{\alpha} \equiv s'}{s \xrightarrow{\alpha} s'}$$

$$\frac{s_1 \xrightarrow{\mathrm{n} \rhd \bar{w}} s_1' \quad s_2 \xrightarrow{\mathrm{n} \lhd \bar{v}} s_2' \quad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \quad \mathrm{noConf}(s_1 \mid s_2, \mathrm{n}, \bar{v}, |\sigma|)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\,\sigma\,|\sigma|\,\bar{v}} s_1' \mid s_2'}$$

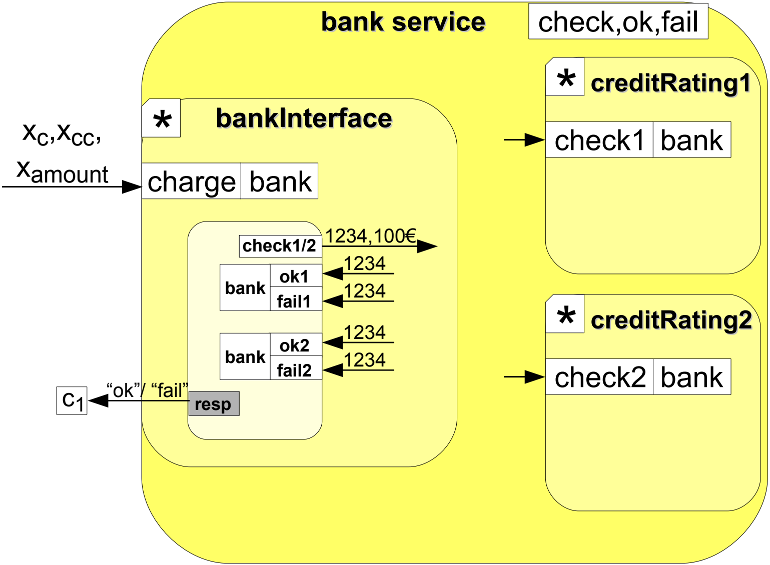
$$\frac{s \xrightarrow{\mathrm{n}\,\sigma \uplus \{x \mapsto v\}\,\ell\,\bar{v}} s'}{[x]\,s \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s' \cdot \{x \mapsto v\}} \qquad \frac{s_1 \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s_1' \qquad \mathrm{noConf}(s_2, \mathrm{n}, \bar{v}, \ell)}{s_1 \mid s_2 \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s_1' \mid s_2}$$

$$\mathbf{kill}(k) \xrightarrow{k} \mathbf{0} \qquad \frac{s \xrightarrow{\alpha} s'}{\{\!|s|\!\} \xrightarrow{\alpha} \{\!|s'|\!\}} \qquad \frac{s_1 \xrightarrow{\alpha} s_1' \quad \alpha \neq k, \mathrm{n}\,\sigma\,\ell\,\bar{v}}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

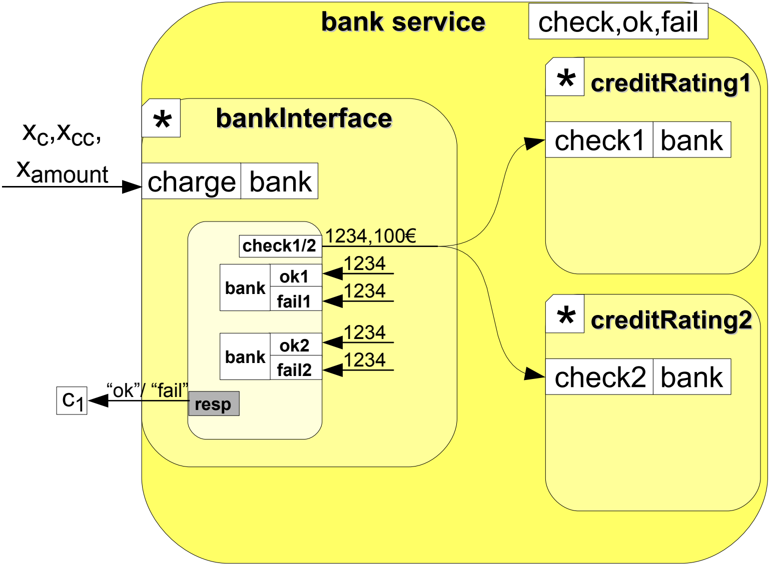
$$\frac{s \xrightarrow{k} s'}{[k]\,s \xrightarrow{\dagger} [k]\,s'} \qquad \frac{s \xrightarrow{k} s' \quad k \neq e}{[e]\,s \xrightarrow{k} [e]\,s'} \qquad \frac{s_1 \xrightarrow{k} s_1'}{s_1 \mid s_2 \xrightarrow{k} s_1' \mid \mathrm{halt}(s_2)}$$

$$\frac{s \xrightarrow{\dagger} s'}{[e]\,s \xrightarrow{\dagger} [e]\,s'} \qquad \frac{s \xrightarrow{\alpha} s' \quad e \notin \mathrm{e}(\alpha) \quad \alpha \neq k, \dagger \quad \mathrm{noKill}(s, e)}{[e]\,s \xrightarrow{\alpha} [e]\,s'}$$

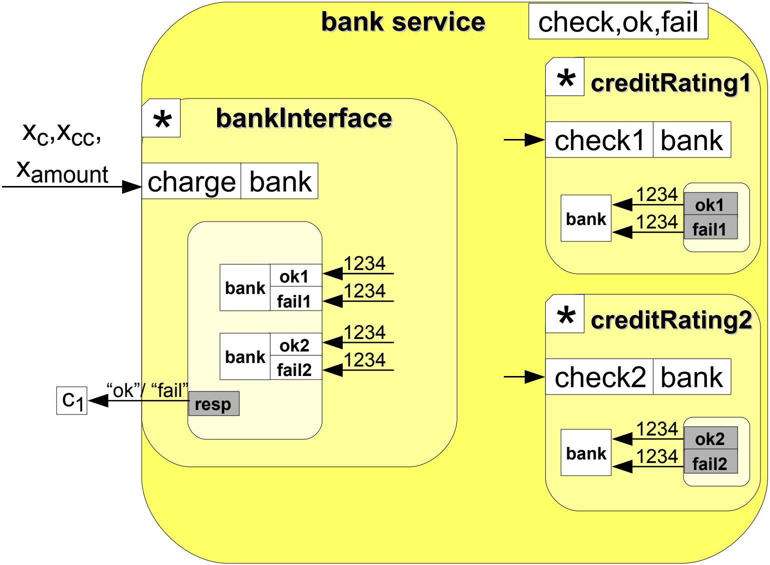
COWS: *multi rating* bank service example



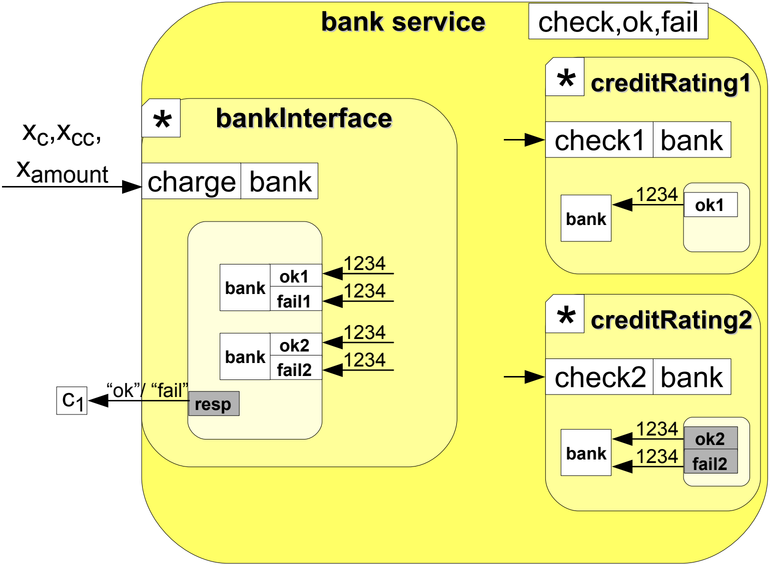
COWS: *multi rating* bank service example



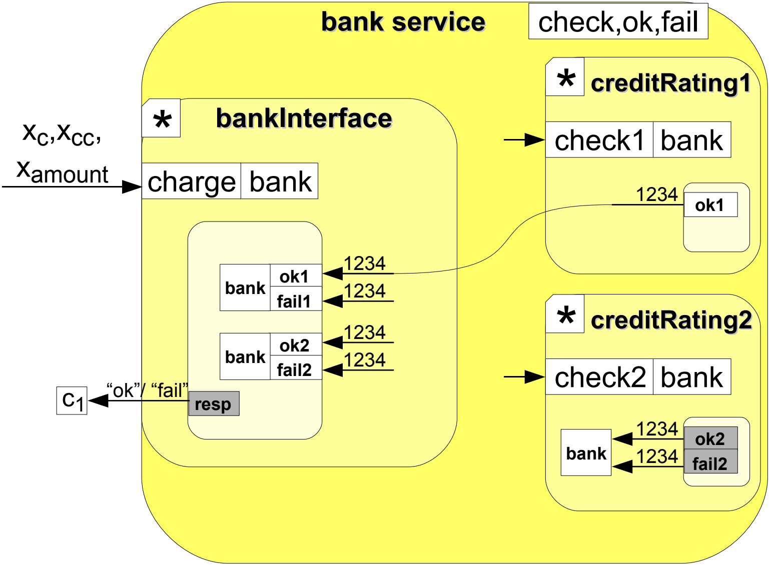
COWS: *multi rating* bank service example



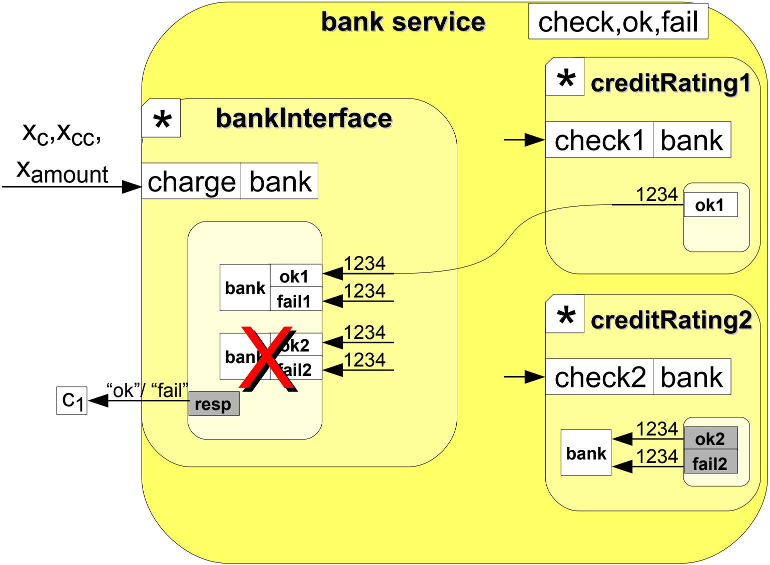
COWS: *multi rating* bank service example



COWS: *multi rating* bank service example



COWS: *multi rating* bank service example



COWS: *multi rating* bank service example

$$
\begin{array}{l}
[\mathrm{check1}, \mathrm{check2}, \mathrm{ok1}, \mathrm{ok2}, \mathrm{fail1}, \mathrm{fail2}] \\
(\, * \,\mathrm{bankInterface} \mid * \,\mathrm{creditRating1} \mid * \,\mathrm{creditRating2}\,)
\end{array}
$$

$$
\begin{array}{l}
\mathrm{bankInterface} \;\triangleq \\
\quad [\mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}] \\
\quad \mathrm{bank} \bullet \mathrm{charge}?\langle \mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}} \rangle . \\
\quad (\, \mathrm{bank} \bullet \mathrm{check1}!\langle \mathrm{x_{cc}}, \mathrm{x_{amount}} \rangle \mid \mathrm{bank} \bullet \mathrm{check2}!\langle \mathrm{x_{cc}}, \mathrm{x_{amount}} \rangle \\
\quad\;\; \mid [k]\, (\, \mathrm{bank} \bullet \mathrm{ok1}?\langle \mathrm{x_{cc}} \rangle . (\, \mathbf{kill}(k) \mid \{\!|\, \mathrm{x_c} \bullet \mathrm{resp}!\langle \text{“ok”} \rangle \,|\!\}\,) \\
\qquad\qquad\quad + \mathrm{bank} \bullet \mathrm{fail1}?\langle \mathrm{x_{cc}} \rangle . s_1 \\
\qquad\qquad \mid \mathrm{bank} \bullet \mathrm{ok2}?\langle \mathrm{x_{cc}} \rangle . (\, \mathbf{kill}(k) \mid \{\!|\, \mathrm{x_c} \bullet \mathrm{resp}!\langle \text{“ok”} \rangle \,|\!\}\,) \\
\qquad\qquad\quad\;\; + \mathrm{bank} \bullet \mathrm{fail2}?\langle \mathrm{x_{cc}} \rangle . s_2 \,)\,)
\end{array}
$$

COWS: peculiar examples

Protected kill activity

- Execution of a kill activity within a protection block

$$[k]\,(\{\!|s_1 \;|\; \{\!|s_2|\!\} \;|\; \mathbf{kill}(k)|\!\} \;|\; s_3) \;|\; s_4 \xrightarrow{\dagger} [k]\,\{\!|s_2|\!\} \;|\; s_4$$

For simplicity, assume that $\mathrm{halt}(s_1) = \mathrm{halt}(s_3) = \mathbf{0}$

- **kill**(k) terminates all parallel services inside delimitation $[k]$ (i.e. s_1 and s_3), except those that are protected at the same nesting level of the kill activity (i.e. s_2)

COWS: peculiar examples

Protected kill activity

- Execution of a kill activity within a protection block

$$[k]\,(\{\!|s_1 \mid \{\!|s_2|\!\} \mid \mathbf{kill}(k)|\!\} \mid s_3) \mid s_4 \xrightarrow{\dagger} [k]\,\{\!|s_2|\!\} \mid s_4$$

For simplicity, assume that $\mathrm{halt}(s_1) = \mathrm{halt}(s_3) = \mathbf{0}$

- **kill**(k) terminates all parallel services inside delimitation $[k]$ (i.e. s_1 and s_3), except those that are protected at the same nesting level of the kill activity (i.e. s_2)

COWS: peculiar examples

Protected kill activity

- Execution of a kill activity within a protection block

$$[k]\,(\{\!|\, s_1 \;|\; \{\!|\, s_2 \,|\!\} \;|\; \mathbf{kill}(k) \,|\!\} \;|\; s_3) \;|\; s_4 \xrightarrow{\dagger} [k]\,\{\!|\, s_2 \,|\!\} \;|\; s_4$$

  For simplicity, assume that $\mathrm{halt}(s_1) = \mathrm{halt}(s_3) = \mathbf{0}$

- **kill**(k) terminates all parallel services inside delimitation $[k]$ (i.e. s_1 and s_3), except those that are protected at the same nesting level of the kill activity (i.e. s_2)

COWS: peculiar examples

Interplay between communication and kill activity

$$p \bullet o!\langle n \rangle \mid [k]\,([x]\,p \bullet o?\langle x \rangle.s \mid \mathbf{kill}(k)) \xrightarrow{\dagger} p \bullet o!\langle n \rangle \mid [k]\,[x]\,\mathbf{0}$$

- Kill activities can break communication
- This is the only possible evolution (kills are executed *eagerly*)
- Communication can be guaranteed by protecting the receive

$$p \bullet o!\langle n \rangle \mid [k]\,([x]\,\{\!|p \bullet o?\langle x \rangle.s|\!\} \mid \mathbf{kill}(k)) \xrightarrow{\dagger}$$

$$p \bullet o!\langle n \rangle \mid [k]\,([x]\,\{\!|p \bullet o?\langle x \rangle.s|\!\}) \xrightarrow{p \bullet o\,\emptyset\,1\,\langle n \rangle} [k]\,\{\!|s \cdot \{x \mapsto n\}|\!\}$$

COWS: peculiar examples

Interplay between communication and kill activity

$$p \bullet o!\langle n\rangle \mid [k]\,([x]\,p \bullet o?\langle x\rangle.s \mid \mathbf{kill}(k)) \xrightarrow{\dagger} p \bullet o!\langle n\rangle \mid [k]\,[x]\,\mathbf{0}$$

- Kill activities can break communication
- This is the only possible evolution (kills are executed *eagerly*)

- Communication can be guaranteed by protecting the receive

$$p \bullet o!\langle n\rangle \mid [k]\,([x]\,\{\!|p \bullet o?\langle x\rangle.s|\!\} \mid \mathbf{kill}(k)) \xrightarrow{\dagger}$$
$$p \bullet o!\langle n\rangle \mid [k]\,([x]\,\{\!|p \bullet o?\langle x\rangle.s|\!\}) \xrightarrow{p \bullet o\,\emptyset\,1\,\langle n\rangle} [k]\,\{\!|s \cdot \{x \mapsto n\}|\!\}$$

COWS expressiveness

Considerations on COWS expressiveness

- Encoding other calculi
  - π-calculus, Localized π-calculus (Lπ), ...
  - SCC (Session Centered Calculus)
  - Orc
  - WS-CALCULUS
  - B*lite* (a lightweight version of WS-BPEL)
- COWS (like other calculi equipped with priority) is not encodable into mainstream calculi (e.g. CCS and π-calculus) [EXPRESS'10]
- Modelling imperative and orchestration constructs
  - Assignment, conditional choice, sequential composition,...
  - WS-BPEL flow graphs, fault and compensation handlers
  - QoS requirement specifications and SLA negotiations [WWV'07]
  - Timed orchestration constructs [ICTAC'07]

Considerations on COWS expressiveness

- Encoding other calculi
  - π-calculus, Localized π-calculus (Lπ), ...
  - SCC (Session Centered Calculus)
  - Orc
  - WS-CALCULUS
  - B*lite* (a lightweight version of WS-BPEL)

- COWS (like other calculi equipped with priority) is not encodable into mainstream calculi (e.g. CCS and π-calculus) [EXPRESS'10]

- Modelling imperative and orchestration constructs
  - Assignment, conditional choice, sequential composition,...
  - WS-BPEL flow graphs, fault and compensation handlers
  - QoS requirement specifications and SLA negotiations [WWV'07]
  - Timed orchestration constructs [ICTAC'07]

Considerations on COWS expressiveness

- Encoding other calculi
  - π-calculus, Localized π-calculus (Lπ), . . .
  - SCC (Session Centered Calculus)
  - Orc
  - WS-CALCULUS
  - B*lite* (a lightweight version of WS-BPEL)

- COWS (like other calculi equipped with priority) is not encodable into mainstream calculi (e.g. CCS and π-calculus) [EXPRESS'10]

- Modelling imperative and orchestration constructs
  - Assignment, conditional choice, sequential composition,. . .
  - WS-BPEL flow graphs, fault and compensation handlers
  - QoS requirement specifications and SLA negotiations [WWV'07]
  - Timed orchestration constructs [ICTAC'07]

Considerations on COWS expressiveness

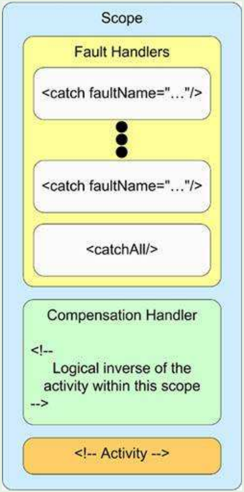
- Encoding other calculi
  - π-calculus, Localized π-calculus (Lπ), ...
  - SCC (Session Centered Calculus)
  - Orc
  - WS-CALCULUS
  - B*lite* (a lightweight version of WS-BPEL)

- COWS (like other calculi equipped with priority) is not encodable into mainstream calculi (e.g. CCS and π-calculus) [EXPRESS'10]

- Modelling imperative and orchestration constructs
  - Assignment, conditional choice, sequential composition,...
  - WS-BPEL flow graphs, fault and compensation handlers
  - QoS requirement specifications and SLA negotiations [WWV'07]
  - Timed orchestration constructs [ICTAC'07]

COWS: fault and compensation handling

Scope



Scope

Fault Handlers

<catch faultName="..."/>

<catch faultName="..."/>

<catchAll/>

Compensation Handler

<!--
Logical inverse of the activity within this scope
-->

<!-- Activity -->

COWS: fault and compensation handling

Syntax for compensation

$$\begin{array}{lll} s ::= & \ldots & (\textit{services}) \\ \mid & \textbf{throw}(\phi) & (\textit{fault generator}) \\ \mid & \textbf{compensate}(i) & (\textit{compensate}) \\ \mid & [s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_i & (\textit{scope}) \end{array}$$

- **throw**(ϕ): rises a fault signal ϕ that triggers execution of s if a construct **catch**(ϕ)$\{s\}$ exists within the same scope
- **compensate**(i): invokes a compensation handler of an inner scope i that has already completed normally (i.e. without faulting)
- $[s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_i$: is uniquely identified by i and groups together a service s (the normal behaviour), an optional list of fault handlers, and a compensation handler s_c

COWS: fault and compensation handling

Encoding

$$
\begin{aligned}
\langle\!\langle [s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_i \rangle\!\rangle_k \ = \ & [\phi_1, \ldots, \phi_n]\ (\ \langle\!\langle \textbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ \ldots \ \mid \langle\!\langle \textbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \\
& \mid [k_i]\ \langle\!\langle s \rangle\!\rangle_{k_i}\ ;\ (\ x_{done} \bullet o_{done}!\langle\rangle \mid [k']\ \{\!|\mathtt{undo}?\langle i \rangle.\langle\!\langle s_c \rangle\!\rangle_{k'} |\!\}\ )\ )
\end{aligned}
$$

$$\langle\!\langle \textbf{catch}(\phi)\{s\} \rangle\!\rangle_k \ = \ \mathtt{throw}?\langle\phi\rangle.[k']\ \langle\!\langle s \rangle\!\rangle_{k'}$$

$$\langle\!\langle \textbf{compensate}(i) \rangle\!\rangle_k \ = \ \mathtt{undo}!\langle i \rangle \mid x_{done} \bullet o_{done}!\langle\rangle$$

$$\langle\!\langle \textbf{throw}(\phi) \rangle\!\rangle_k \ = \ \{\!|\mathtt{throw}!\langle\phi\rangle|\!\}\ \mid \textbf{kill}(k)$$

- s_c is installed when the normal behaviour s successfully completes, but it is activated only when signal $\mathtt{undo}!\langle i \rangle$ occurs
- Whenever a fault ϕ occurs, a signal $\mathtt{throw}!\langle\phi\rangle$ triggers execution of the corresponding fault handler (if any), while installed compensation handlers are protected from killing by means of $\{\!|\_|\!\}$
- ‘;’ denotes sequential composition, that can be easily encoded in COWS alike in CCS

COWS: fault and compensation handling

Encoding

$$\begin{aligned}\langle\!\langle [s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_i \rangle\!\rangle_k \; = \;& [\phi_1, \ldots, \phi_n]\;(\,\langle\!\langle \mathbf{catch}(\phi_1)\{s_1\}\rangle\!\rangle_k \mid \ldots \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\}\rangle\!\rangle_k \\ & \mid [k_i]\,\langle\!\langle s \rangle\!\rangle_{k_i}\,;(\,x_{done} \bullet o_{done}!\langle\rangle \mid [k']\,\{\!|\,\texttt{undo?}\langle i\rangle.\langle\!\langle s_c\rangle\!\rangle_{k'}\,|\!\}\,)\,)\end{aligned}$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\}\rangle\!\rangle_k = \texttt{throw?}\langle\phi\rangle.[k']\,\langle\!\langle s\rangle\!\rangle_{k'}$$

$$\langle\!\langle \mathbf{compensate}(i)\rangle\!\rangle_k = \texttt{undo!}\langle i\rangle \mid x_{done} \bullet o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi)\rangle\!\rangle_k = \{\!|\,\texttt{throw!}\langle\phi\rangle\,|\!\} \mid \mathbf{kill}(k)$$

- s_c is installed when the normal behaviour s successfully completes, but it is activated only when signal $\texttt{undo!}\langle i\rangle$ occurs
- Whenever a fault ϕ occurs, a signal $\texttt{throw!}\langle\phi\rangle$ triggers execution of the corresponding fault handler (if any), while installed compensation handlers are protected from killing by means of $\{\!|\_|\!\}$
- ‘;’ denotes sequential composition, that can be easily encoded in COWS alike in CCS

COWS: fault and compensation handling

Encoding

$$
\begin{aligned}
\langle\!\langle [s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_i \rangle\!\rangle_k \; = \;& [\phi_1, \ldots, \phi_n] \, (\, \langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ \ldots \ \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \\
& \mid [k_i] \, \langle\!\langle s \rangle\!\rangle_{k_i} \, ; (\, x_{done} \bullet o_{done}!\langle\rangle \mid [k'] \, \{\!|\mathtt{undo}?\langle i \rangle . \langle\!\langle s_c \rangle\!\rangle_{k'} |\!\} \,)\,)
\end{aligned}
$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k \; = \; \mathtt{throw}?\langle \phi \rangle . [k'] \, \langle\!\langle s \rangle\!\rangle_{k'}$$

$$\langle\!\langle \mathbf{compensate}(i) \rangle\!\rangle_k \; = \; \mathtt{undo}!\langle i \rangle \mid x_{done} \bullet o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k \; = \; \{\!|\mathtt{throw}!\langle \phi \rangle|\!\} \mid \mathbf{kill}(k)$$

- s_c is installed when the normal behaviour s successfully completes, but it is activated only when signal $\mathtt{undo}!\langle i \rangle$ occurs
- Whenever a fault ϕ occurs, a signal $\mathtt{throw}!\langle \phi \rangle$ triggers execution of the corresponding fault handler (if any), while installed compensation handlers are protected from killing by means of $\{\!|\_|\!\}$
- ‘;’ denotes sequential composition, that can be easily encoded in COWS alike in CCS

COWS: fault and compensation handling

Encoding

$$\begin{aligned}\langle\!\langle [s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_i \rangle\!\rangle_k \;=\;& [\phi_1,\ldots,\phi_n]\,(\,\langle\!\langle \textbf{catch}(\phi_1)\{s_1\}\rangle\!\rangle_k \mid \;\ldots\; \mid \langle\!\langle \textbf{catch}(\phi_n)\{s_n\}\rangle\!\rangle_k \\ & \mid [k_i]\,\langle\!\langle s\rangle\!\rangle_{k_i} \,;\, (\,x_{done}\bullet o_{done}!\langle\rangle \mid [k']\,\{\!|\texttt{undo}?\langle i\rangle.\langle\!\langle s_c\rangle\!\rangle_{k'}|\!\}\,)\,)\end{aligned}$$

$$\langle\!\langle \textbf{catch}(\phi)\{s\}\rangle\!\rangle_k = \texttt{throw}?\langle\phi\rangle.[k']\,\langle\!\langle s\rangle\!\rangle_{k'}$$

$$\langle\!\langle \textbf{compensate}(i)\rangle\!\rangle_k = \texttt{undo}!\langle i\rangle \mid x_{done}\bullet o_{done}!\langle\rangle$$

$$\langle\!\langle \textbf{throw}(\phi)\rangle\!\rangle_k = \{\!|\texttt{throw}!\langle\phi\rangle|\!\} \mid \textbf{kill}(k)$$

- s_c is installed when the normal behaviour s successfully completes, but it is activated only when signal $\texttt{undo}!\langle i\rangle$ occurs
- Whenever a fault ϕ occurs, a signal $\texttt{throw}!\langle\phi\rangle$ triggers execution of the corresponding fault handler (if any), while installed compensation handlers are protected from killing by means of $\{\!|\_|\!\}$
- ‘;’ denotes sequential composition, that can be easily encoded in COWS alike in CCS

Considerations on COWS expressiveness

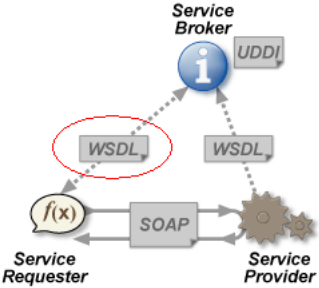
- Encoding other calculi
  - π-calculus, Localized π-calculus (Lπ), . . .
  - SCC (Session Centered Calculus)
  - Orc
  - WS-CALCULUS
  - B*lite* (a lightweight version of WS-BPEL)
- COWS (like other calculi equipped with priority) is not encodable into mainstream calculi (e.g. CCS and π-calculus) [EXPRESS'10]
- Modelling imperative and orchestration constructs
  - Assignment, conditional choice, sequential composition,. . .
  - WS-BPEL flow graphs, fault and compensation handlers
  - QoS requirement specifications and SLA negotiations [WWV'07]
  - Timed orchestration constructs [ICTAC'07]

QoS specifications and SLA negotiations

- We have demonstrated so far that COWS can model service specification, orchestration, reconfiguration, and execution
- Now, we focus on other important phases of the life cycle of service-oriented applications
- We want now to show that service publication, discovery and SLA negotiation can be naturally modelled in COWS
- We exploit ‘constraints’ and operations on them

Discovery and Negotiation

- In a SOA, requesters should be able to *discover* functionalities



- Service descriptions should include both functional and non-functional aspects (*Quality of Service*, QoS)
- Dynamic service discovery can rely on a negotiation mechanism
  - negotiation allows two or more parties to reach a joint agreement about cost and quality of a service
  - the outcome is a contract, called *Service Level Agreement* (SLA)

Dealing with QoS and SLA in COWS

- COWS syntax and semantics are parametrically defined w.r.t.:
  - the set of manipulable values
  - the syntax of expressions
  - the definition of the pattern-matching function

- We appropriately specialize these parameters
  - expressions also include *constraints* and *constraint multisets*
  - additional rules tailor pattern-matching to operations on constraints

- The specialized language combines basic features of *name-passing calculi* and of *concurrent constraint programming*
  - SLA requirements are constraints that can be dynamically generated and composed
  - At the end of the negotiation process, if the resulting set of constraints is consistent then it represents the reached agreement

Dealing with QoS and SLA in COWS

- COWS syntax and semantics are parametrically defined w.r.t.:
  - the set of manipulable values
  - the syntax of expressions
  - the definition of the pattern-matching function
- We appropriately specialize these parameters
  - expressions also include *constraints* and *constraint multisets*
  - additional rules tailor pattern-matching to operations on constraints
- The specialized language combines basic features of *name-passing calculi* and of *concurrent constraint programming*
  - SLA requirements are constraints that can be dynamically generated and composed
  - At the end of the negotiation process, if the resulting set of constraints is consistent then it represents the reached agreement

Dealing with QoS and SLA in COWS

- COWS syntax and semantics are parametrically defined w.r.t.:
  - the set of manipulable values
  - the syntax of expressions
  - the definition of the pattern-matching function

- We appropriately specialize these parameters
  - expressions also include *constraints* and *constraint multisets*
  - additional rules tailor pattern-matching to operations on constraints

- The specialized language combines basic features of *name-passing calculi* and of *concurrent constraint programming*
  - SLA requirements are constraints that can be dynamically generated and composed
  - At the end of the negotiation process, if the resulting set of constraints is consistent then it represents the reached agreement

A credit card issue scenario

client

needs a credit card

bank

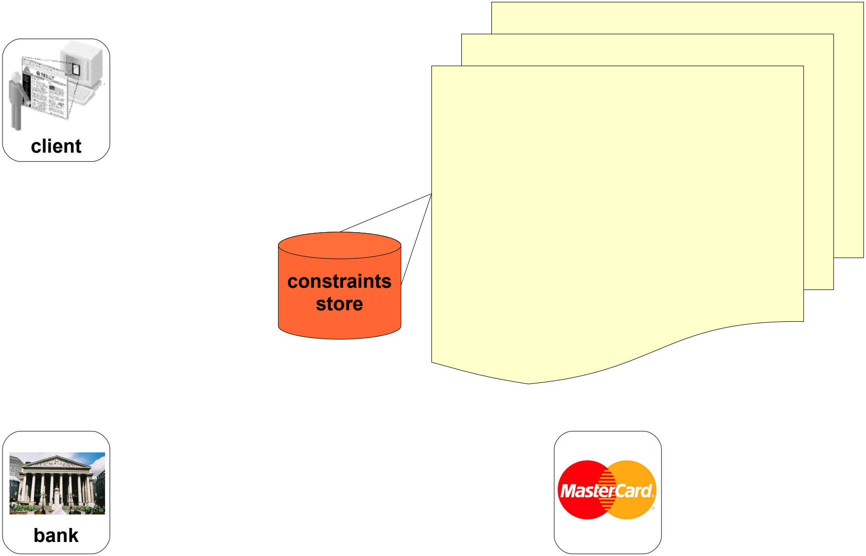
offers different credit cards varying in cost x year and cost x operation

provide credit card services to banks

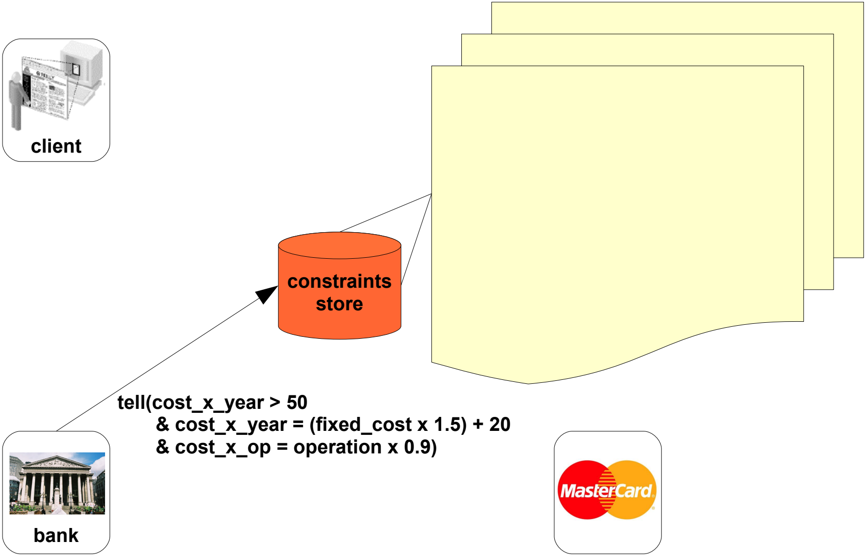
constraints store

allows clients, banks and credit card associations to execute the negotiation process

A credit card issue scenario: negotiation with MC

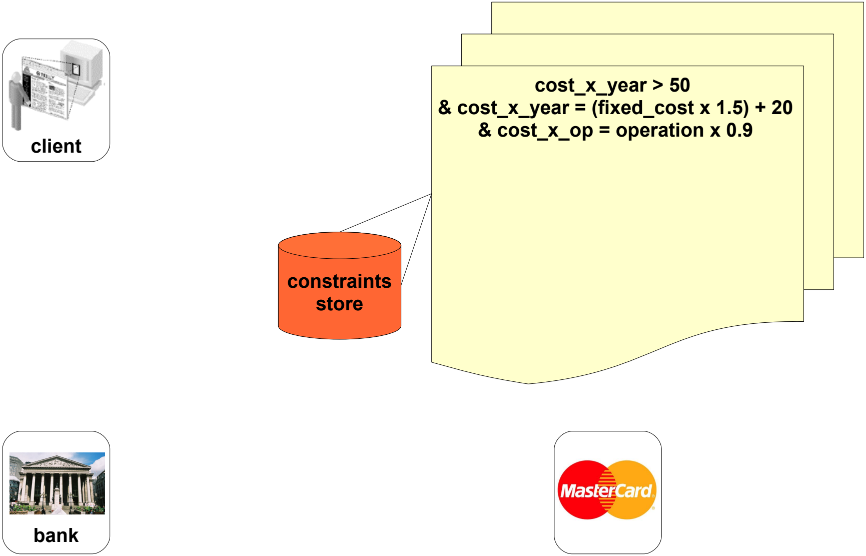


A credit card issue scenario: negotiation with MC



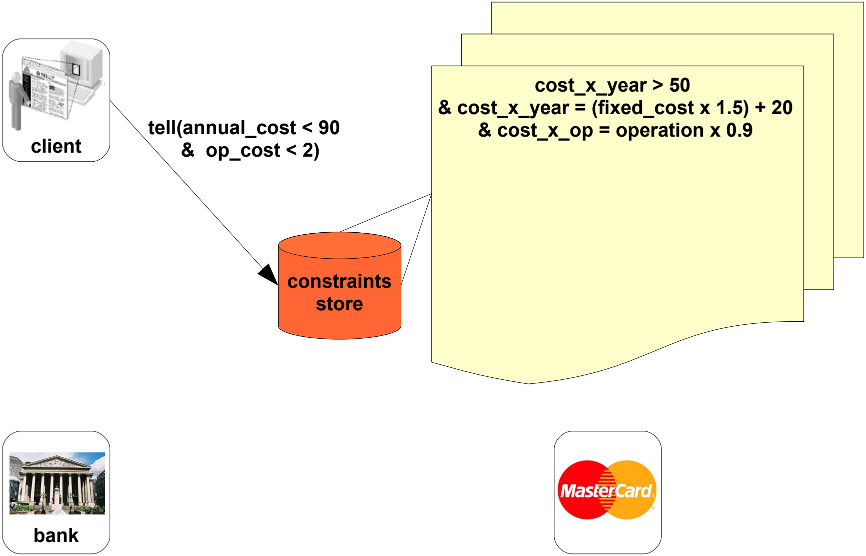
```
tell(cost_x_year > 50
      & cost_x_year = (fixed_cost x 1.5) + 20
      & cost_x_op = operation x 0.9)
```

A credit card issue scenario: negotiation with MC



cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

A credit card issue scenario: negotiation with MC



client

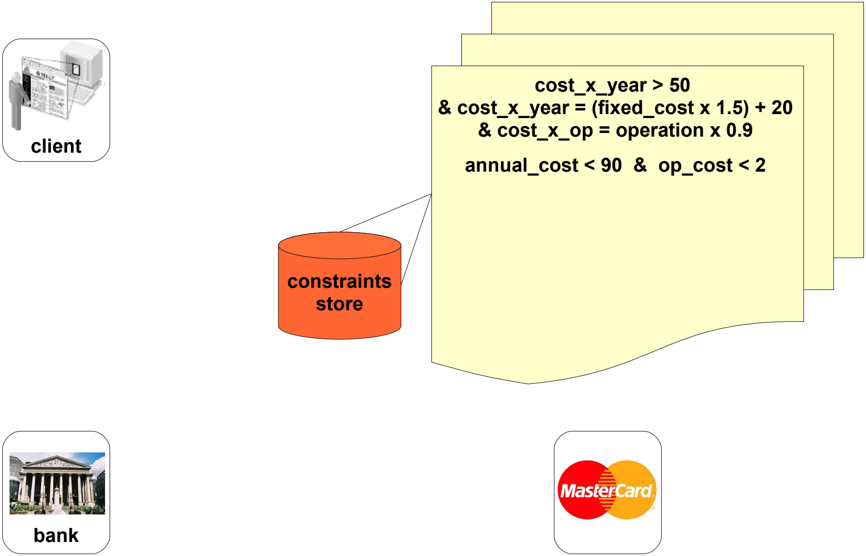
tell(annual_cost < 90
& op_cost < 2)

constraints store

cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

bank

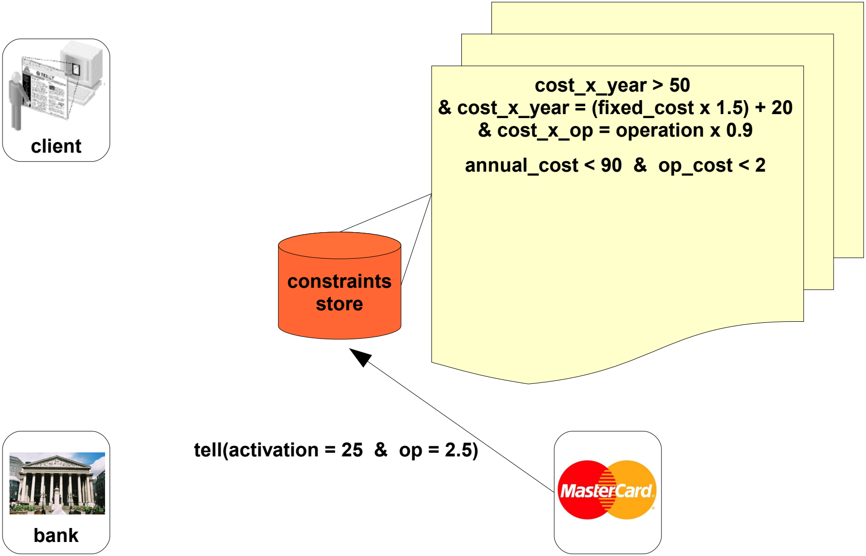
A credit card issue scenario: negotiation with MC



cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

A credit card issue scenario: negotiation with MC



client

constraints store

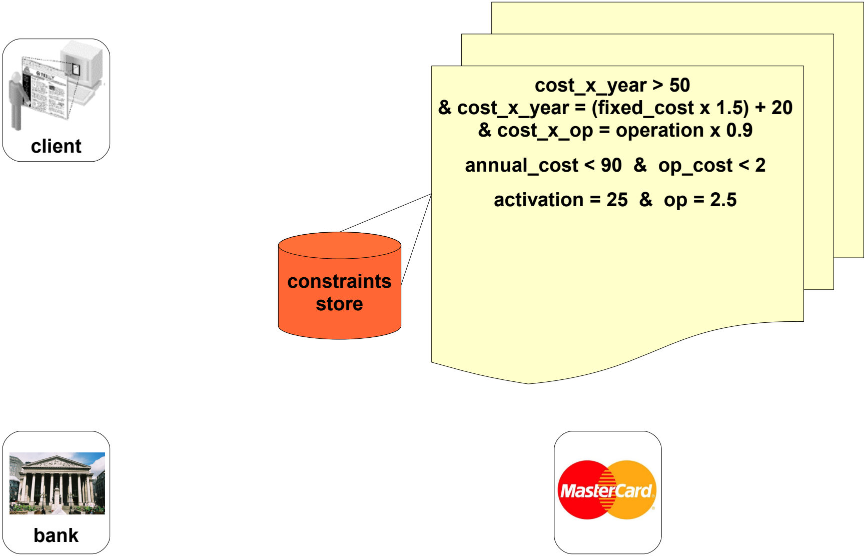
cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

tell(activation = 25 & op = 2.5)

bank

A credit card issue scenario: negotiation with MC



client

cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

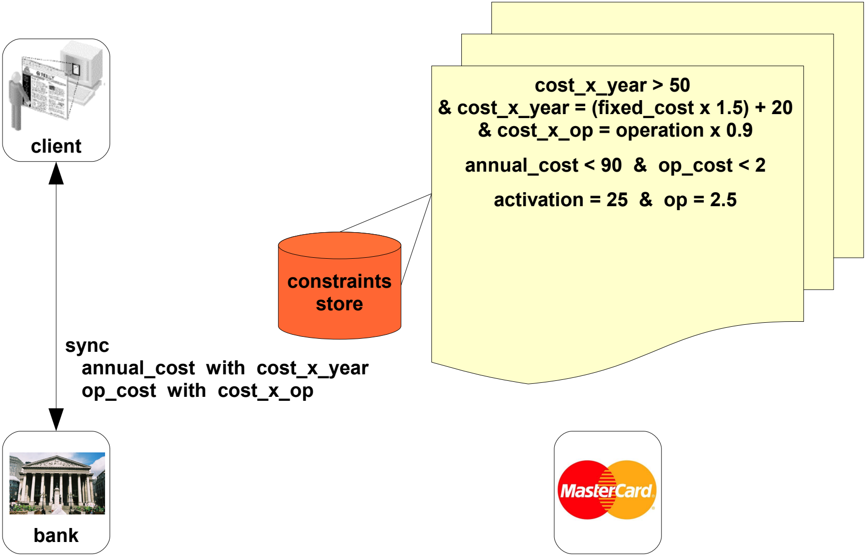
annual_cost < 90 & op_cost < 2

activation = 25 & op = 2.5

constraints store

bank

A credit card issue scenario: negotiation with MC



client

constraints store

cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

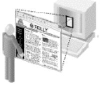
annual_cost < 90 & op_cost < 2

activation = 25 & op = 2.5

sync
annual_cost with cost_x_year
op_cost with cost_x_op

bank

A credit card issue scenario: negotiation with MC



client

constraints store

cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

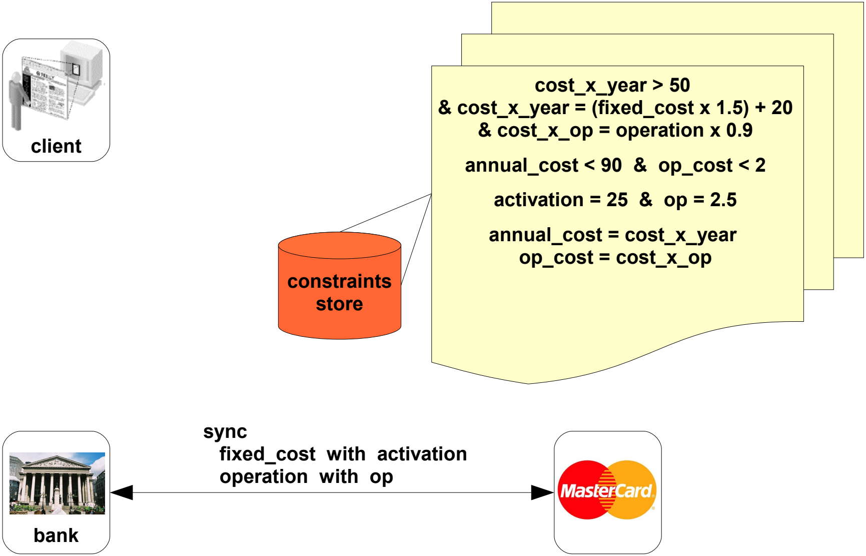
activation = 25 & op = 2.5

annual_cost = cost_x_year
op_cost = cost_x_op

bank

MasterCard

A credit card issue scenario: negotiation with MC



client

cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

activation = 25 & op = 2.5

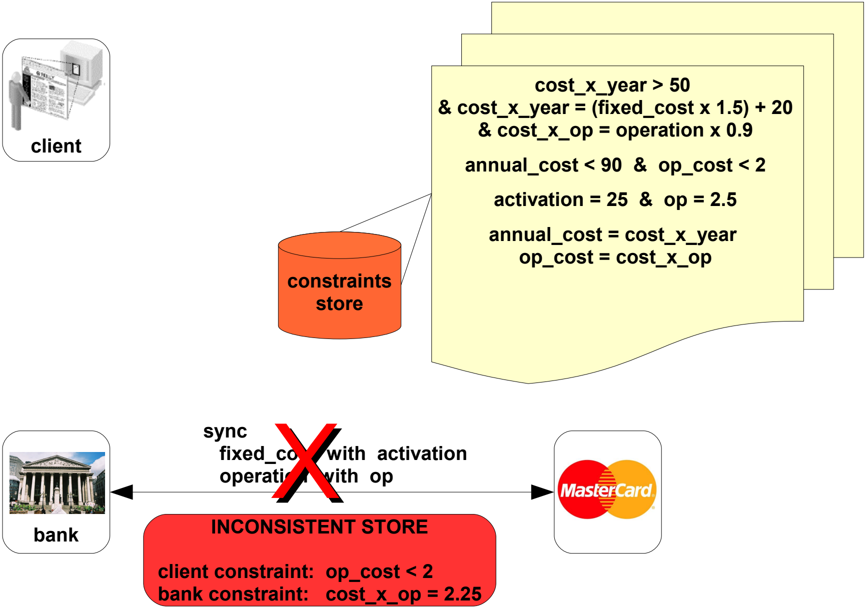
annual_cost = cost_x_year
op_cost = cost_x_op

constraints store

sync
fixed_cost with activation
operation with op

bank

A credit card issue scenario: negotiation with MC



client

cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

activation = 25 & op = 2.5

annual_cost = cost_x_year
op_cost = cost_x_op

constraints store

bank

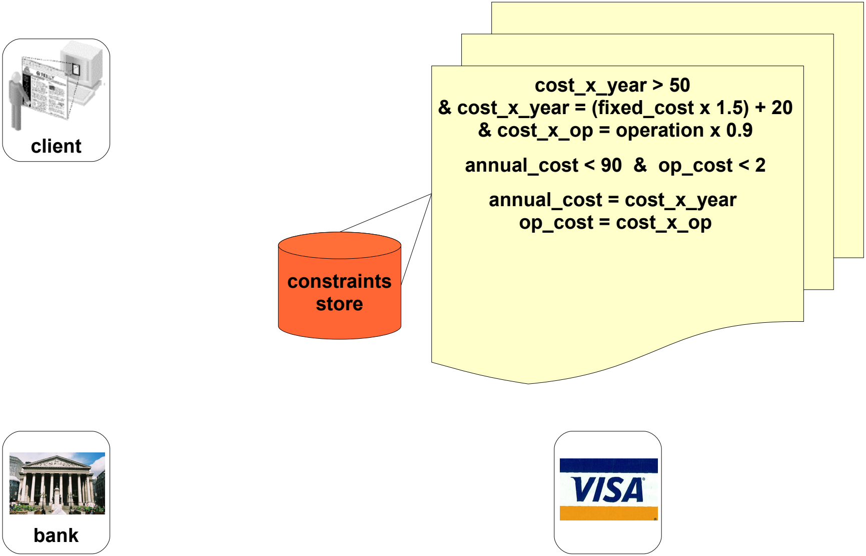
sync
fixed_co with activation
operati with op

MasterCard

INCONSISTENT STORE

client constraint: op_cost < 2
bank constraint: cost_x_op = 2.25

A credit card issue scenario: negotiation with VISA

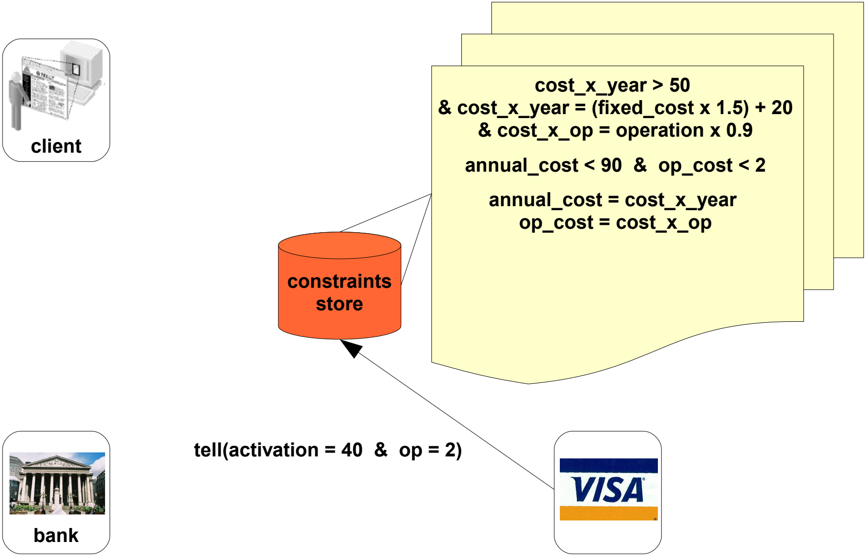


cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

annual_cost = cost_x_year
op_cost = cost_x_op

A credit card issue scenario: negotiation with VISA



client

cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

annual_cost = cost_x_year
op_cost = cost_x_op

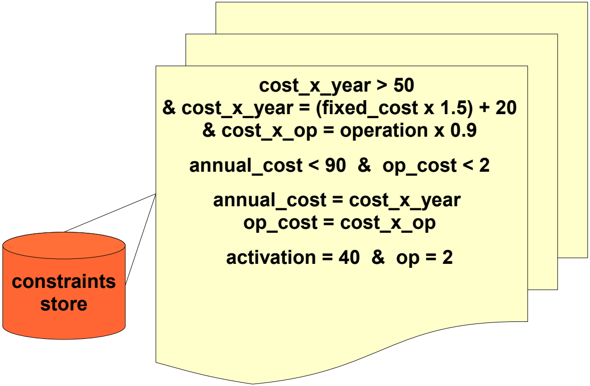
constraints store

tell(activation = 40 & op = 2)

bank

A credit card issue scenario: negotiation with VISA

client



cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

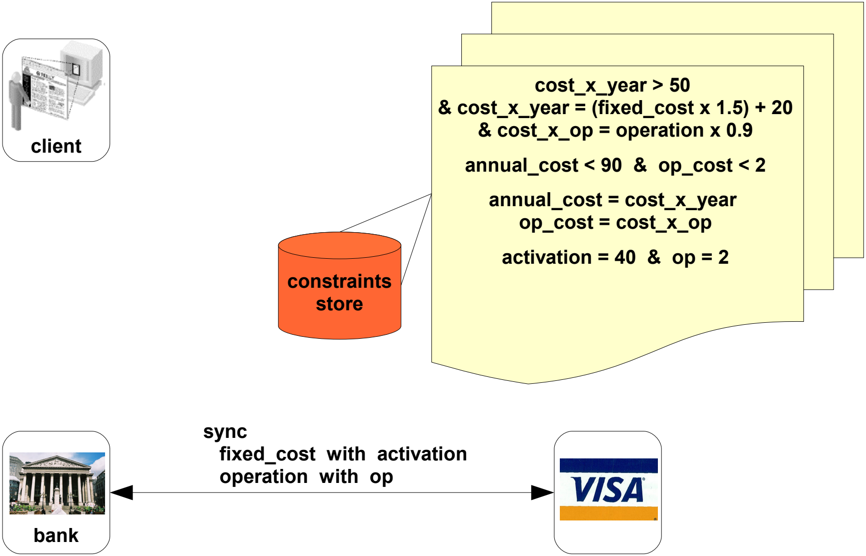
annual_cost = cost_x_year
op_cost = cost_x_op

activation = 40 & op = 2

bank

VISA

A credit card issue scenario: negotiation with VISA



client

constraints store

cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

annual_cost < 90 & op_cost < 2

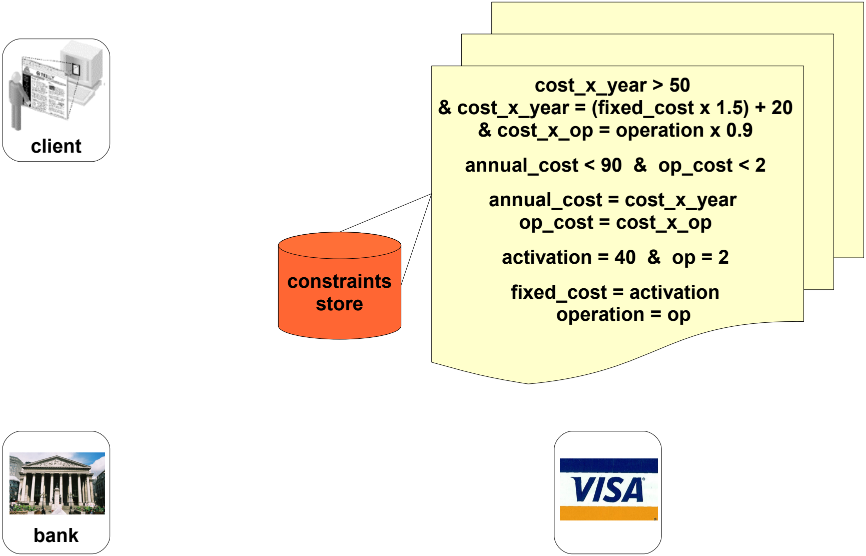
annual_cost = cost_x_year
op_cost = cost_x_op

activation = 40 & op = 2

sync
fixed_cost with activation
operation with op

bank

A credit card issue scenario: negotiation with VISA



cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

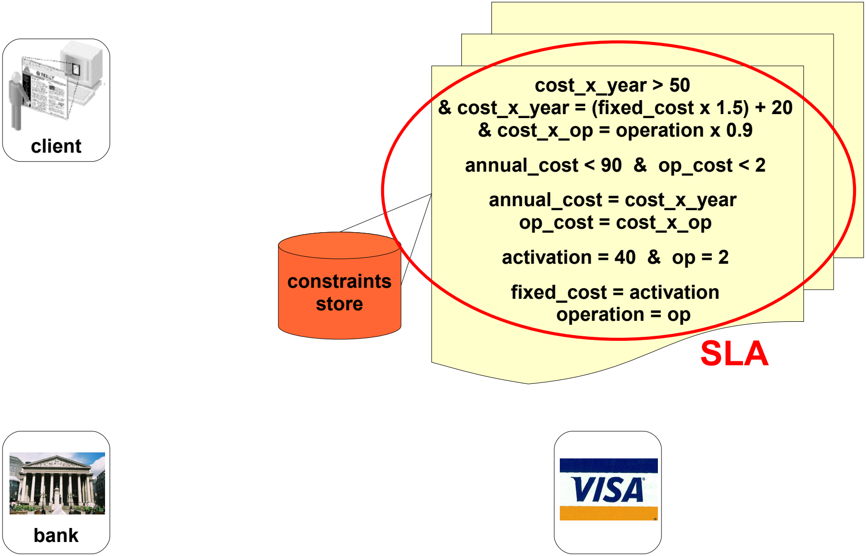
annual_cost < 90 & op_cost < 2

annual_cost = cost_x_year
op_cost = cost_x_op

activation = 40 & op = 2

fixed_cost = activation
operation = op

A credit card issue scenario: negotiation with VISA



cost_x_year > 50
& cost_x_year = (fixed_cost x 1.5) + 20
& cost_x_op = operation x 0.9

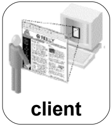
annual_cost < 90 & op_cost < 2

annual_cost = cost_x_year
op_cost = cost_x_op

activation = 40 & op = 2

fixed_cost = activation
operation = op

A credit card issue scenario: COWS specification



$[\mathtt{annual\_cost}, \mathtt{op\_cost}]$
$\mathtt{tell}(\mathtt{annual\_cost} < 90 \wedge \mathtt{op\_cost} < 2).$
$\mathsf{bank} \bullet \mathsf{sync}!\langle \mathtt{annual\_cost}, \mathtt{op\_cost} \rangle$

$[\mathtt{cost\_x\_year}, \mathtt{cost\_x\_op}, \mathtt{fixed\_cost}, \mathtt{operation}]$
$\mathtt{tell}(\mathtt{cost\_x\_year} > 50$
$\quad \wedge\ \mathtt{cost\_x\_year} = (\mathtt{fixed\_cost} \times 1.5) + 20$
$\quad \wedge\ \mathtt{cost\_x\_op} = \mathtt{operation} \times 0.9).$
$\mathsf{bank} \bullet \mathsf{sync}?\langle \mathtt{cost\_x\_year}, \mathtt{cost\_x\_op} \rangle.$
$\mathsf{visa} \bullet \mathsf{sync}!\langle \mathtt{fixed\_cost}, \mathtt{operation} \rangle$

$[\mathtt{activation}, \mathtt{op}]$
$\mathtt{tell}(\mathtt{activation} = 40 \wedge \mathtt{op} = 2).$
$\mathsf{visa} \bullet \mathsf{sync}?\langle \mathtt{activation}, \mathtt{op} \rangle$

Using COWS for concurrent constraint programming

- A shared store of constraints provides partial information about possible values of variables

The shared store

$$store_C \;\triangleq\; [\hat{n}]\,(\,\hat{n}!\langle C\rangle \mid *\,[x]\,\hat{n}?\langle x\rangle.(\,p_s\bullet o_{get}!\langle x\rangle \mid [y]\,p_s\bullet o_{set}?\langle y\rangle.\hat{n}!\langle y\rangle\,)\,)$$

- Services can act on the store by performing operations

Add/remove constraints to/from the store

$$\langle\!\langle \texttt{tell}\ c.s\rangle\!\rangle = [\hat{n}]\,(\,\hat{n}!\langle c\rangle \mid [y]\,\hat{n}?\langle y\rangle.[x]\,p_s\bullet o_{get}?\langle\langle y,x\rangle\rangle.(\{\!|\,p_s\bullet o_{set}!\langle x \uplus \{y\}\rangle\,|\!\} \mid \langle\!\langle s\rangle\!\rangle)\,)$$

$$\langle\!\langle \texttt{retract}\ c.s\rangle\!\rangle = [\hat{n}]\,(\,\hat{n}!\langle c\rangle \mid [y]\,\hat{n}?\langle y\rangle.[x]\,p_s\bullet o_{get}?\langle x\rangle.(\{\!|\,p_s\bullet o_{set}!\langle x - y\rangle\,|\!\} \mid \langle\!\langle s\rangle\!\rangle)\,)$$

Check entailment/consistency of a constraint by/with the store

$$\langle\!\langle \texttt{ask}\ c.s\rangle\!\rangle = [\hat{n}]\,(\,\hat{n}!\langle c^{\vdash}\rangle \mid [y]\,\hat{n}?\langle y\rangle.[x]\,p_s\bullet o_{get}?\langle\langle y,x\rangle\rangle.(\{\!|\,p_s\bullet o_{set}!\langle x\rangle\,|\!\} \mid \langle\!\langle s\rangle\!\rangle)\,)$$

$$\langle\!\langle \texttt{check}\ c.s\rangle\!\rangle = [\hat{n}]\,(\,\hat{n}!\langle c\rangle \mid [y]\,\hat{n}?\langle y\rangle.[x]\,p_s\bullet o_{get}?\langle\langle y,x\rangle\rangle.(\{\!|\,p_s\bullet o_{set}!\langle x\rangle\,|\!\} \mid \langle\!\langle s\rangle\!\rangle)\,)$$

Considerations on COWS expressiveness

- Encoding other calculi
  - π-calculus, Localized π-calculus (Lπ), ...
  - SCC (Session Centered Calculus)
  - Orc
  - WS-CALCULUS
  - B*lite* (a lightweight version of WS-BPEL)

- COWS (like other calculi equipped with priority) is not encodable into mainstream calculi (e.g. CCS and π-calculus) [EXPRESS'10]

- Modelling imperative and orchestration constructs
  - Assignment, conditional choice, sequential composition,...
  - WS-BPEL flow graphs, fault and compensation handlers
  - QoS requirement specifications and SLA negotiations [WWV'07]
  - Timed orchestration constructs [ICTAC'07]

Dealing with timed orchestration constructs

- *Timed activities* are frequently exploited in service orchestration for dealing with service transactions or with message losses
  - A service could wait a callback message for a certain amount of time after which, if no callback has been received, it invokes another operation or throws a fault

- Timed activities can be exploited to allow services not to get stuck forever waiting on a receive

- It is not known to what extent timed computation can be reduced to untimed forms of computation

- C◷WS extends COWS with timed orchestration constructs

Dealing with timed orchestration constructs

- *Timed activities* are frequently exploited in service orchestration for dealing with service transactions or with message losses
  - A service could wait a callback message for a certain amount of time after which, if no callback has been received, it invokes another operation or throws a fault

- Timed activities can be exploited to allow services not to get stuck forever waiting on a receive

- It is not known to what extent timed computation can be reduced to untimed forms of computation

- C◷WS extends COWS with timed orchestration constructs

Dealing with timed orchestration constructs

- *Timed activities* are frequently exploited in service orchestration for dealing with service transactions or with message losses
  - A service could wait a callback message for a certain amount of time after which, if no callback has been received, it invokes another operation or throws a fault

- Timed activities can be exploited to allow services not to get stuck forever waiting on a receive

- It is not known to what extent timed computation can be reduced to untimed forms of computation

- C◷WS extends COWS with timed orchestration constructs

C◷WS: a timed extension of COWS

COWS (Calculus for Orchestration of Web Services)

μCOWS

- Communication activities
- Parallel composition
- Replication
- Choice
- Delimitation

Termination constructs

- Kill activity
- Protection

C🕘WS: a timed extension of COWS

COWS (Calculus for Orchestration of Web Services)

μCOWS

- Communication activities
- Parallel composition
- Replication
- Choice
- Delimitation

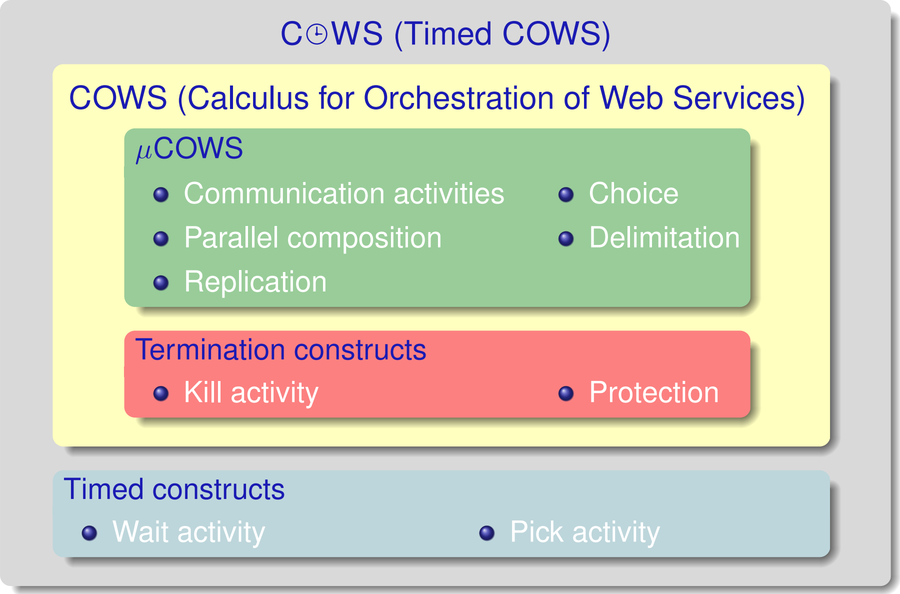
Termination constructs

- Kill activity
- Protection

Timed constructs

- Wait activity
- Pick activity

C🕒WS: a timed extension of COWS



C🕒WS (Timed COWS)

COWS (Calculus for Orchestration of Web Services)

μCOWS

- Communication activities
- Parallel composition
- Replication
- Choice
- Delimitation

Termination constructs

- Kill activity
- Protection

Timed constructs

- Wait activity
- Pick activity

Syntax of C🕒WS

$$\begin{array}{lll}
s ::= & & \text{(services)} \\
 & \mathbf{kill}(k) & \text{(kill)} \\
 \mid & u \bullet u'!\bar{e} & \text{(invoke)} \\
 \mid & \sum_{i=0}^{r} g_i . s_i & \text{(choice/pick)} \\
 \mid & s \mid s & \text{(parallel composition)} \\
 \mid & \{\!| s |\!\} & \text{(protection)} \\
 \mid & [e]\, s & \text{(delimitation)} \\
 \mid & * \, s & \text{(replication)} \\
 & & \\
g ::= p \bullet o?\bar{w} & \mid \ \text{🕒}_{e} & \text{(guards)}
\end{array}$$

(notations)
k: *(killer) labels*
e: *expressions*
  x: *variables*
  v: *values*
    n, p, o: *names*
    δ: *time intervals*
u: *vars* | *names*
w: *vars* | *values*
e: *labels* | *vars* | *names*

- *Time intervals* δ are positive numbers
- The *wait activity* 🕒_{e} specifies the time interval, whose value is given by evaluation of e, the executing service has to wait for
- Time elapsing cannot make a choice within a *pick* activity $\sum$, while the occurrence of a timeout can

C🕘WS operational semantics

COWS operational semantics rules

+

$$\mathbf{0} \xrightarrow{\delta} \mathbf{0} \qquad * s \xrightarrow{\delta} * s \qquad u \bullet u'!\bar{e} \xrightarrow{\delta} u \bullet u'!\bar{e}$$

$$\frac{s \xrightarrow{\delta} s'}{\{|s|\} \xrightarrow{\delta} \{|s'|\}} \qquad \frac{s \xrightarrow{\delta} s'}{[e]\, s \xrightarrow{\delta} [e]\, s'} \qquad p \bullet o?\bar{w}.s \xrightarrow{\delta} p \bullet o?\bar{w}.s$$

$$\text{🕘}_{0}.s \xrightarrow{\dagger} s \qquad \frac{[\![e]\!] \neq \delta'}{\text{🕘}_{e}.s \xrightarrow{\delta} \text{🕘}_{e}.s} \qquad \frac{\delta \leqslant [\![e]\!]}{\text{🕘}_{e}.s \xrightarrow{\delta} \text{🕘}_{[\![e-\delta]\!]}.s}$$

$$\frac{g_1 \xrightarrow{\delta} g_1' \qquad g_2 \xrightarrow{\delta} g_2'}{g_1 + g_2 \xrightarrow{\delta} g_1' + g_2'} \qquad \frac{s_1 \xrightarrow{\delta} s_1' \qquad s_2 \xrightarrow{\delta} s_2'}{s_1 \mid s_2 \xrightarrow{\delta} s_1' \mid s_2'}$$

C🕒WS: bank service with timeout

$$[\mathrm{check}, \mathrm{ok}, \mathrm{fail}]\,(\, *\,\mathrm{bankTimedInterface} \mid *\,\mathrm{creditRating}\,)$$

$$\begin{array}{lcl}
\mathrm{bankTimedInterface} & \triangleq & [x_c, x_{cc}, x_{amount}] \\
 & & \mathrm{bank} \bullet \mathrm{charge}?\langle x_c, x_{cc}, x_{amount}\rangle . \\
 & & (\,\mathrm{bank} \bullet \mathrm{check}!\langle x_{cc}, x_{amount}\rangle \\
 & & \quad \mid \mathrm{bank} \bullet \mathrm{ok}?\langle x_{cc}\rangle .\, x_c \bullet \mathrm{resp}!\langle \text{“ok”} \rangle \\
 & & \qquad + \mathrm{bank} \bullet \mathrm{fail}?\langle x_{cc}\rangle .\, x_c \bullet \mathrm{resp}!\langle \text{“fail”} \rangle \\
 & & \qquad + \text{🕒}_{120} .\, x_c \bullet \mathrm{resp}!\langle \text{“timeout”} \rangle \,)
\end{array}$$

The bank service guarantees a response within two minutes

C🕒WS: bank service with timeout

$$
\begin{array}{lcl}
\mathrm{bank} \bullet \mathrm{charge}!\langle \mathrm{c}, 1234, 100\text{€}\rangle & & [\mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}] \\
\mid [\mathrm{x}]\,(\mathrm{c} \bullet \mathrm{resp}?\langle \mathrm{x}\rangle.\mathrm{s} \mid \mathrm{s}') & \mid & \mathrm{bank} \bullet \mathrm{charge}?\langle \mathrm{x_c}, \mathrm{x_{cc}}, \mathrm{x_{amount}}\rangle. \\
 & & (\,\mathrm{bank} \bullet \mathrm{check}!\langle \mathrm{x_{cc}}, \mathrm{x_{amount}}\rangle \\
 & & \;\mid \mathrm{bank} \bullet \mathrm{ok}?\langle \mathrm{x_{cc}}\rangle.\, \mathrm{x_c} \bullet \mathrm{resp}!\langle \text{“ok”} \rangle \\
 & & \quad + \mathrm{bank} \bullet \mathrm{fail}?\langle \mathrm{x_{cc}}\rangle.\, \mathrm{x_c} \bullet \mathrm{resp}!\langle \text{“fail”} \rangle \\
 & & \quad + \text{🕒}_{120}.\, \mathrm{x_c} \bullet \mathrm{resp}!\langle \text{“timeout”} \rangle\,)
\end{array}
$$

C🕒WS: bank service with timeout

$$\begin{array}{lcl}
\mathrm{bank} \bullet \mathrm{charge}!\langle c, 1234, 100€\rangle & & [x_c, x_{cc}, x_{amount}] \\
\mid [x]\,(c \bullet \mathrm{resp}?\langle x\rangle.s \mid s') & \mid & \mathrm{bank} \bullet \mathrm{charge}?\langle x_c, x_{cc}, x_{amount}\rangle. \\
 & & (\,\mathrm{bank} \bullet \mathrm{check}!\langle x_{cc}, x_{amount}\rangle \\
 & & \;\mid \mathrm{bank} \bullet \mathrm{ok}?\langle x_{cc}\rangle.\, x_c \bullet \mathrm{resp}!\langle \text{“ok”}\rangle \\
 & & \quad + \mathrm{bank} \bullet \mathrm{fail}?\langle x_{cc}\rangle.\, x_c \bullet \mathrm{resp}!\langle \text{“fail”}\rangle \\
 & & \quad + \text{🕒}_{120}.\, x_c \bullet \mathrm{resp}!\langle \text{“timeout”}\rangle\,)
\end{array}$$

The client sends its request to the bank service …

C🕒WS: bank service with timeout

$$\begin{array}{lll}
[x]\,(\mathrm{c} \bullet \mathrm{resp}?\langle \mathrm{x}\rangle.\mathrm{s} \mid \mathrm{s}') & \quad\mid\quad & \mathrm{bank} \bullet \mathrm{check}!\langle 1234, 100\text{€}\rangle \\
 & & \;\mid \mathrm{bank} \bullet \mathrm{ok}?\langle 1234\rangle.\, \mathrm{c} \bullet \mathrm{resp}!\langle \text{“ok”}\rangle \\
 & & \quad + \mathrm{bank} \bullet \mathrm{fail}?\langle 1234\rangle.\, \mathrm{c} \bullet \mathrm{resp}!\langle \text{“fail”}\rangle \\
 & & \quad + \text{🕒}_{120}.\, \mathrm{c} \bullet \mathrm{resp}!\langle \text{“timeout”}\rangle
\end{array}$$

C🕒WS: bank service with timeout

$$
\begin{array}{lcl}
[\mathrm{x}]\,(\mathrm{c} \bullet \mathrm{resp}?\langle \mathrm{x} \rangle.\mathrm{s} \mid \mathrm{s}') & \mid & \mathrm{bank} \bullet \mathrm{check}!\langle 1234, 100\text{€} \rangle \\
 & & \quad \mid \mathrm{bank} \bullet \mathrm{ok}?\langle 1234 \rangle .\, \mathrm{c} \bullet \mathrm{resp}!\langle \text{“ok”} \rangle \\
 & & \qquad + \mathrm{bank} \bullet \mathrm{fail}?\langle 1234 \rangle .\, \mathrm{c} \bullet \mathrm{resp}!\langle \text{“fail”} \rangle \\
 & & \qquad + \text{🕒}_{120} .\, \mathrm{c} \bullet \mathrm{resp}!\langle \text{“timeout”} \rangle
\end{array}
$$

The credit rating service is unavailable and a first minute elapses . . .

C🕒WS: bank service with timeout

$$
\begin{array}{lcl}
[x]\,(c \bullet resp?\langle x \rangle.s \mid s') & \mid & bank \bullet check!\langle 1234, 100\text{€} \rangle \\
 & & \quad \mid bank \bullet ok?\langle 1234 \rangle .\, c \bullet resp!\langle \text{“ok”} \rangle \\
 & & \qquad + \, bank \bullet fail?\langle 1234 \rangle .\, c \bullet resp!\langle \text{“fail”} \rangle \\
 & & \qquad + \, \text{🕒}_{60} .\, c \bullet resp!\langle \text{“timeout”} \rangle
\end{array}
$$

C🕒WS: bank service with timeout

$$
\begin{array}{lll}
[x]\,(c \bullet resp?\langle x\rangle.s \mid s') & \mid & bank \bullet check!\langle 1234, 100€\rangle \\
 & & \quad \mid bank \bullet ok?\langle 1234\rangle.\, c \bullet resp!\langle \text{“ok”}\rangle \\
 & & \qquad + bank \bullet fail?\langle 1234\rangle.\, c \bullet resp!\langle \text{“fail”}\rangle \\
 & & \qquad + \text{🕒}_{60}.\, c \bullet resp!\langle \text{“timeout”}\rangle
\end{array}
$$

The last minute elapses …

C🕒WS: bank service with timeout

$$
\begin{array}{lcl}
[\mathrm{x}]\,(\mathrm{c}\bullet\mathrm{resp}?\langle \mathrm{x}\rangle.\mathrm{s} \mid \mathrm{s}') & \mid & \mathrm{bank}\bullet\mathrm{check}!\langle 1234, 100€\rangle \\
 & & \quad\mid \mathrm{bank}\bullet\mathrm{ok}?\langle 1234\rangle.\,\mathrm{c}\bullet\mathrm{resp}!\langle\text{“ok”}\rangle \\
 & & \qquad + \mathrm{bank}\bullet\mathrm{fail}?\langle 1234\rangle.\,\mathrm{c}\bullet\mathrm{resp}!\langle\text{“fail”}\rangle \\
 & & \qquad + \text{🕒}_{0}.\,\mathrm{c}\bullet\mathrm{resp}!\langle\text{“timeout”}\rangle
\end{array}
$$

C🕒WS: bank service with timeout

$$
\begin{array}{lcl}
[\mathsf{x}]\,(\mathsf{c}\bullet\mathsf{resp}?\langle\mathsf{x}\rangle.\mathsf{s} \mid \mathsf{s}') & \quad\mid\quad & \mathsf{bank}\bullet\mathsf{check}!\langle 1234, 100\text{€}\rangle \\
 & & \quad\mid \mathsf{bank}\bullet\mathsf{ok}?\langle 1234\rangle.\,\mathsf{c}\bullet\mathsf{resp}!\langle\text{“ok”}\rangle \\
 & & \qquad + \mathsf{bank}\bullet\mathsf{fail}?\langle 1234\rangle.\,\mathsf{c}\bullet\mathsf{resp}!\langle\text{“fail”}\rangle \\
 & & \qquad + \text{🕒}_{0}.\,\mathsf{c}\bullet\mathsf{resp}!\langle\text{“timeout”}\rangle
\end{array}
$$

The timeout expires . . .

this way the receives along bank • ok and bank • fail are discarded

C🕒WS: bank service with timeout

$$
\begin{array}{lcl}
[\mathrm{x}]\,(\mathrm{c}\bullet\mathrm{resp}?\langle\mathrm{x}\rangle.\mathrm{s} \mid \mathrm{s}') & \mid & \mathrm{bank}\bullet\mathrm{check}!\langle 1234, 100€\rangle \\
 & & \quad \mid \mathrm{c}\bullet\mathrm{resp}!\langle\,\text{“timeout”}\,\rangle
\end{array}
$$

C🕒WS: bank service with timeout

$$[x]\,(\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}\rangle.\mathsf{s} \mid \mathsf{s}') \qquad \mid \qquad \begin{array}{l}\mathsf{bank} \bullet \mathsf{check}!\langle 1234, 100€\rangle \\ \mid \mathsf{c} \bullet \mathsf{resp}!\langle \text{“timeout”} \rangle\end{array}$$

The client is notified that the time is expired . . .

so he can try to contact again the bank or call an assistance service

Service engines

- Time passes synchronously for all services in parallel
  - all services run on a same service *engine*
  - the services share the same clock and can be tightly coupled
- Existing SOC systems are loosely coupled
  - usually deployed on top of distributed systems that offer only weak guarantees on the upper bound of inter-location clock drift

Example: client & compound bank service

$\text{client} \mid [\text{check}, \text{ok}, \text{fail}]\,(\, * \,\text{bankInterface} \mid * \,\text{creditRating}\,)$

- client and bank are loosely coupled and can be deployed on different engines
- The subservices bankInterface and creditRating are tightly coupled and must be colocated
- We introduce explicitly the notions of *service engine* and of *deployment* of services on engines

$\{\,\text{client}\,\} \mid \{\,[\text{check}, \text{ok}, \text{fail}]\,(\, * \,\text{bankInterface} \mid * \,\text{creditRating}\,)\,\}$

Service engines

- Time passes synchronously for all services in parallel
  - all services run on a same service *engine*
  - the services share the same clock and can be tightly coupled
- Existing SOC systems are loosely coupled
  - usually deployed on top of distributed systems that offer only weak guarantees on the upper bound of inter-location clock drift

Example: client & compound bank service

$\mathrm{client} \mid [\mathrm{check}, \mathrm{ok}, \mathrm{fail}]\,(\,*\,\mathrm{bankInterface} \mid *\,\mathrm{creditRating}\,)$

- client and bank are loosely coupled and can be deployed on different engines
- The subservices bankInterface and creditRating are tightly coupled and must be colocated
- We introduce explicitly the notions of *service engine* and of *deployment* of services on engines

$\{\,\mathrm{client}\,\} \mid \{\,[\mathrm{check}, \mathrm{ok}, \mathrm{fail}]\,(\,*\,\mathrm{bankInterface} \mid *\,\mathrm{creditRating}\,)\,\}$

Service engines

- C⌚WS is extended by introducing the category of *engines*

$$\mathbb{E} ::= 0 \mid \{s\} \mid [n]\mathbb{E} \mid \mathbb{E} \mid \mathbb{E}$$

- Communication can take place *intra-engine* or *inter-engine*
- Passing of time is modelled
  - *synchronously* for services deployed on the same service engine
  - *asynchronously* among different engines

References

References 1/4

A WSDL-based type system for WS-BPEL
A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of COORDINATION'06, LNCS 4038, 2006.

A calculus for orchestration of web services
A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of ESOP'07, LNCS 4421, 2007.
▸ go back

Regulating data exchange in service oriented applications
A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of FSEN'07, LNCS 4767, 2007.
▸ go back

COWS: A timed service-oriented calculus
A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of ICTAC'07, LNCS 4711, 2007. ▸ go back

Stochastic COWS
D. Prandi, P. Quaglia. Proc. of ICSOC'07, LNCS 4749, 2007.

References 2/4

- A model checking approach for verifying COWS specifications
  A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, F. Tiezzi.
  Proc. of FASE'08, LNCS 4961, 2008. ▸ go back

- Service discovery and negotiation with COWS
  A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of WWV'07, ENTCS 200(3), 2008. ▸ go back

- Specifying and Analysing SOC Applications with COWS
  A. Lapadula, R. Pugliese, F. Tiezzi. In Concurrency, Graphs and Models, LNCS 5065, 2008.

- SENSORIA Patterns: Augmenting Service Engineering with Formal Analysis, Transformation and Dynamicity
  M. Wirsing, et al. Proc. of ISOLA'08, Communications in Computer and Information Science 17, 2008.

- A formal account of WS-BPEL
  A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of COORDINATION'08, LNCS 5052, 2008.

References 3/4

Formal analysis of BPMN via a translation into COWS
D. Prandi, P. Quaglia, N. Zannone. Proc. of COORDINATION’08, LNCS 5052, 2008.

Relational Analysis of Correlation
J. Bauer, F. Nielson, H.R. Nielson, H. Pilegaard. Proc. of SAS’08, LNCS 5079, 2008.

A Symbolic Semantics for a Calculus for Service-Oriented Computing
R. Pugliese, F. Tiezzi, N. Yoshida. Proc. of PLACES’08, ENTCS 241, 2009.

Specification and analysis of SOC systems using COWS: A finance case study
F. Banti, A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of WWV’08, ENTCS 235(C), 2009.

From Architectural to Behavioural Specification of Services
L. Bocchi, J.L. Fiadeiro, A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of FESCA’09, ENTCS 253/1, 2009.

References 4/4

- On observing dynamic prioritised actions in SOC
  R. Pugliese, F. Tiezzi, N. Yoshida. Proc. of ICALP'09, LNCS 5556, 2009.
  ▸ go back

- On secure implementation of an IHE XUA-based protocol for authenticating healthcare professionals
  M. Masi, R. Pugliese, F. Tiezzi. Proc. of ICISS'09, LNCS 5905, 2009.

- Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing
  M. Wirsing and M. Hölzl Editors. LNCS, 2010. To appear.

- An Accessible Verification Environment for UML Models of Services
  F. Banti, R. Pugliese, F. Tiezzi. Journal of Symbolic Computation, 2010. To appear.

- A criterion for separating process calculi
  F. Banti, R. Pugliese, F. Tiezzi. Proc. of EXPRESS'10, 2010. ▸ go back