Specifying and analisying SOC applications with COWS

– Applications in orchestration of web services –

Rosario Pugliese and Francesco Tiezzi

Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze

SEFM School 2010
"Advanced applications of model-checking techniques"

Pisa, Italy - September 7th, 2010

About this talk

Title

Specifying and analisying SOC applications with COWS

Domain

Software engineering methodologies for service-oriented applications

Outline

- Scenario and motivations
- A gentle introduction to COWS
- COWS expressiveness
- Analysis techniques for COWS specifications
- Concluding remarks and future work
- This afternoon: stochastic extension (by Paola Quaglia)

About this talk

Title

Specifying and analisying SOC applications with COWS

Domain

Software engineering methodologies for service-oriented applications

Outline

- Scenario and motivations
- A gentle introduction to COWS
- COWS expressiveness
- Analysis techniques for COWS specifications
- Concluding remarks and future work
- This afternoon: stochastic extension (by Paola Quaglia)

About this talk

Title

Specifying and analisying SOC applications with COWS

Domain

Software engineering methodologies for service-oriented applications

Outline

- Scenario and motivations
- A gentle introduction to COWS
- COWS expressiveness
- Analysis techniques for COWS specifications
- Concluding remarks and future work
- This afternoon: stochastic extension (by Paola Quaglia)

About this talk

Title

Specifying and analisying SOC applications with COWS

Domain

Software engineering methodologies for service-oriented applications

Outline

- Scenario and motivations
- A gentle introduction to COWS
- COWS expressiveness
- Analysis techniques for COWS specifications
- Concluding remarks and future work
- This afternoon: stochastic extension (by Paola Quaglia)

About this talk

Title

Specifying and analisying SOC applications with COWS

Domain

Software engineering methodologies for service-oriented applications

Outline

- Scenario and motivations
- A gentle introduction to COWS
- COWS expressiveness
- Analysis techniques for COWS specifications
- Concluding remarks and future work
- This afternoon: stochastic extension (by Paola Quaglia)

About this talk

Title

Specifying and analisying SOC applications with COWS

Domain

Software engineering methodologies for service-oriented applications

Outline

- Scenario and motivations
- A gentle introduction to COWS
- COWS expressiveness
- Analysis techniques for COWS specifications
- Concluding remarks and future work
- This afternoon: stochastic extension (by Paola Quaglia)

About this talk

Title

Specifying and analisying SOC applications with COWS

Domain

Software engineering methodologies for service-oriented applications

Outline

- Scenario and motivations
- A gentle introduction to COWS
- COWS expressiveness
- Analysis techniques for COWS specifications
- Concluding remarks and future work
- This afternoon: stochastic extension (by Paola Quaglia)

Scenario and motivations

Service-Oriented Computing (SOC)

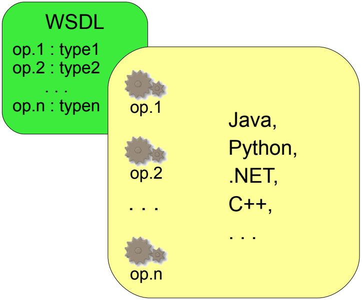
- An emerging paradigm for distributed and e-business computing
- Finds its origin in object-oriented and component-based software development
- Aims at enabling developers to build networks of integrated and collaborative applications, regardless of
    - the platform where the applications run (e.g., the operating system)
    - the programming language used to develop them

  through the use of loosely coupled, reusable software components
- A modern attempt to cope with old problems related to information interchange, software integration, and B2B
- Many instantiations: e.g. grid computing and Web Services

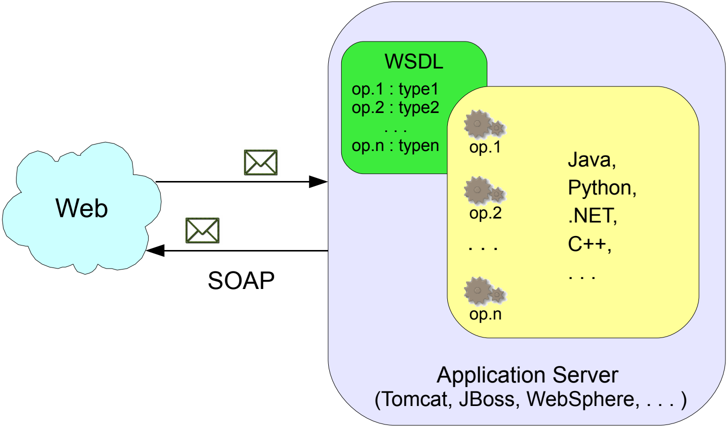
Web Services

- Make available the functionalities that a company wants to expose over the Web, so that they can be exploited by other services
- Their underlying architecture is the World Wide Web
  - Widespread and extensively used platform
  - Suitable to connect different companies and customers
- Independently developed applications can be
  - exposed as services
  - interconnected by exploiting the Web infrastructure and the relative standards, e.g. HTTP, XML, SOAP, WSDL and UDDI
- Facilitate automated integration of newly built and legacy applications, both within and across organizational boundaries

Web Services



Web Services



Web Services Composition

- XML-based technologies like WSDL, UDDI and SOAP
  - permit describing, locating and invoking web services
  - are usually sufficient for simple B2B application integration needs

- Creation of complex B2B applications and automated integration of business processes across enterprises require managing such features as
  - asynchronous interactions
  - concurrency
  - workflow coordination
  - business transaction activities and exceptions

  ...which the above mentioned standards do not deal with

- This raises the need for designing and employing

  Web Services composition languages,

  an additional layer on top of the Web Services protocol stack

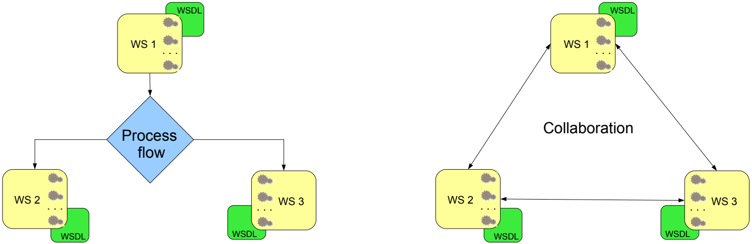
Orchestration vs. Choreography

- Service composition permits to build complex services out of simpler ones and is still an open challenge
- There are two main views of web services composition
  - Orchestration (= Executeable Process)
    - Description of web services interactions, including the business logic and execution order of the interactions
    - Interactions may span applications and/or organizations, and result in a long-lived, transactional process
    - The process is always controlled from the perspective of *one* of the business parties
    - Main enabling technology: WS-BPEL (OASIS standard)
  - Choreography (= Multi-party Collaboration)
    - Description of the externally observable message exchanges between *multiple* web services
    - No party truly ‘owns’ the conversation
    - More collaborative in nature: each party involved in the process describes the role it plays in each interaction
    - Main enabling technology: WS-CDL (W3C Recommendation)

Orchestration vs. Choreography

- Service composition permits to build complex services out of simpler ones and is still an open challenge

- There are two main views of web services composition



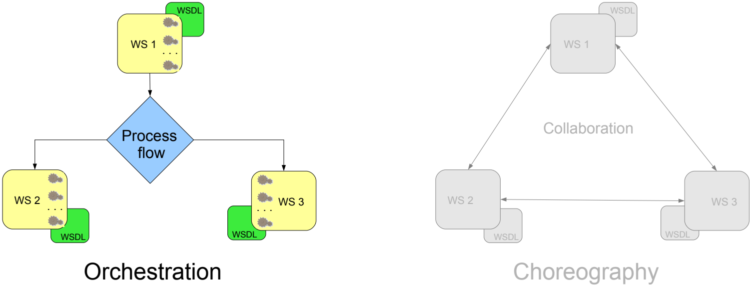
Orchestration

Choreography

We focus on web service orchestration

Orchestration vs. Choreography

- Service composition permits to build complex services out of simpler ones and is still an open challenge
- There are two main views of web services composition



We focus on web service orchestration

Services & Business processes

- A process *orchestrating* web services is called *business process* i.e. an active entity that invokes available services according to a given set of rules to meet some business requirements

- A business process specifies
  - the potential execution order of operations originating from a collection of Web Services
  - the shared data passed between these services
  - the trading partners that are involved in the joint process
  - their roles with respect to the process
  - joint exception handling conditions for the collection of Web Services

  and other factors that may influence how Web Services or organizations participate in a process

- Web service orchestration thus permits to program complex inter-enterprise workflow tasks and business transactions

Services & Business processes

- A process *orchestrating* web services is called *business process* i.e. an active entity that invokes available services according to a given set of rules to meet some business requirements

- A business process specifies
  - the potential execution order of operations originating from a collection of Web Services
  - the shared data passed between these services
  - the trading partners that are involved in the joint process
  - their roles with respect to the process
  - joint exception handling conditions for the collection of Web Services

  and other factors that may influence how Web Services or organizations participate in a process

- Web service orchestration thus permits to program complex inter-enterprise workflow tasks and business transactions

Services & Business processes

- A process *orchestrating* web services is called *business process* i.e. an active entity that invokes available services according to a given set of rules to meet some business requirements

- A business process specifies
  - the potential execution order of operations originating from a collection of Web Services
  - the shared data passed between these services
  - the trading partners that are involved in the joint process
  - their roles with respect to the process
  - joint exception handling conditions for the collection of Web Services

  and other factors that may influence how Web Services or organizations participate in a process

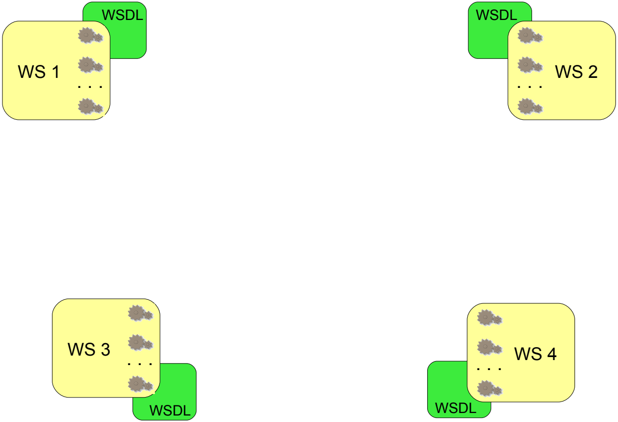
- Web service orchestration thus permits to program complex inter-enterprise workflow tasks and business transactions

Services & Business processes

- Business processes can be exposed as web services

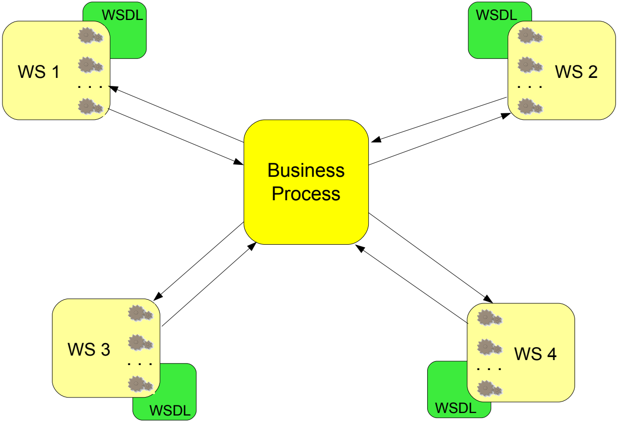
Services & Business processes

- Business processes can be exposed as web services



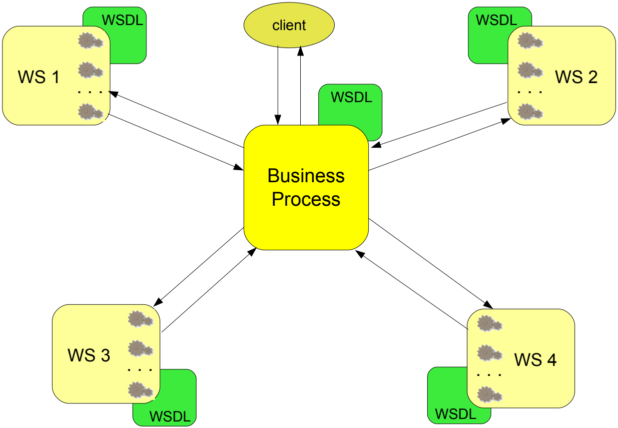
Services & Business processes

- Business processes can be exposed as web services



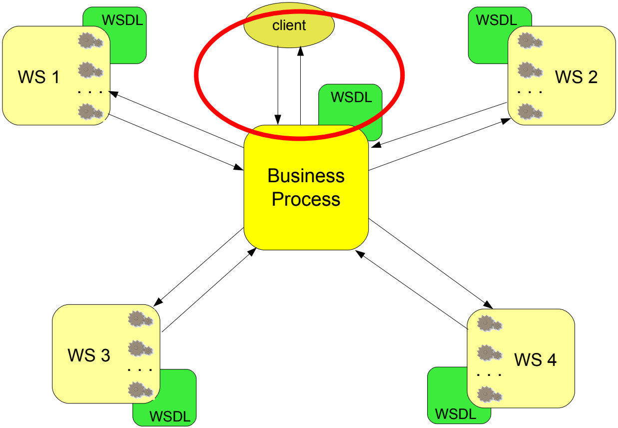
Services & Business processes

- Business processes can be exposed as web services



Services & Business processes

- Business processes can be exposed as web services



Instantiation & Message Correlation

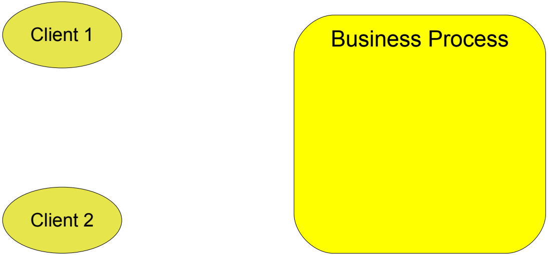
- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)

Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)



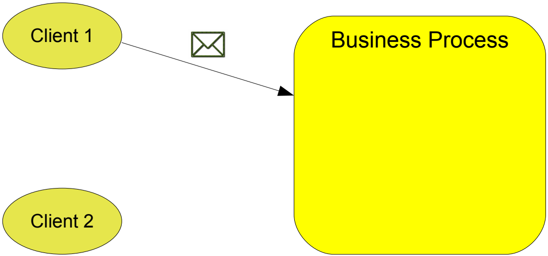
Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created

- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)

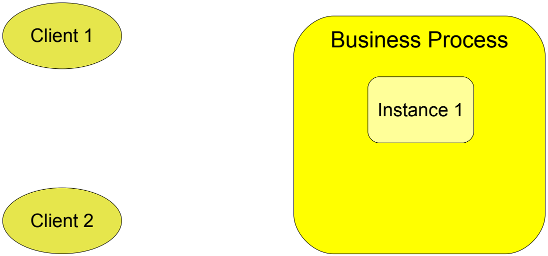


Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)



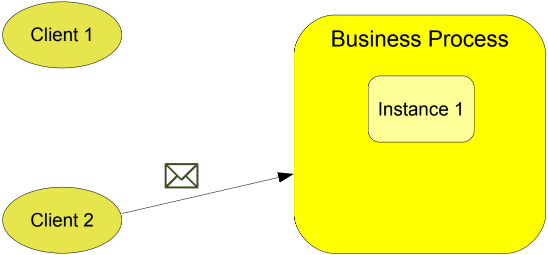
Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created

- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)

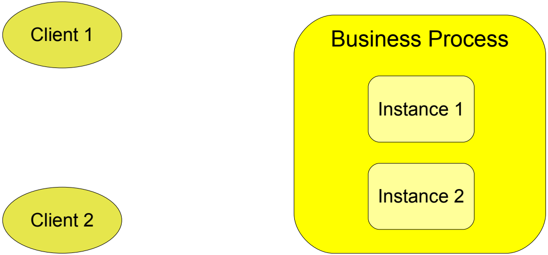


Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)



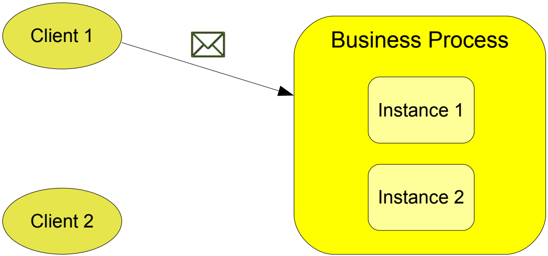
Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created

- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)

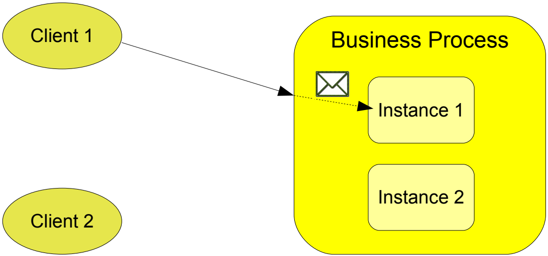


Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)

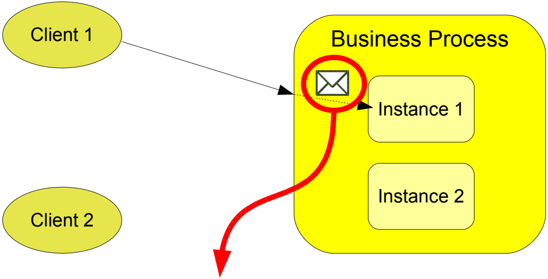


Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)

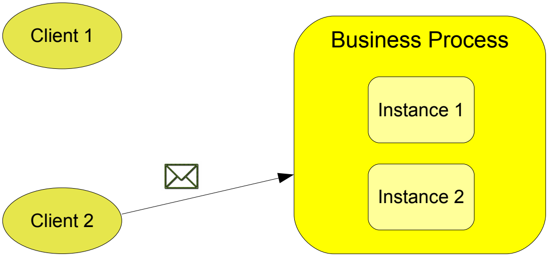


Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)

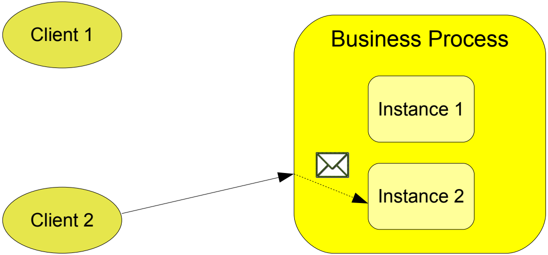


Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)

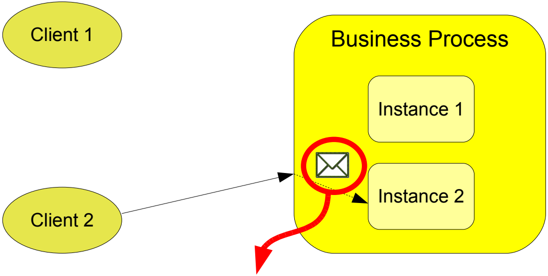


Message correlation

The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)



Message correlation

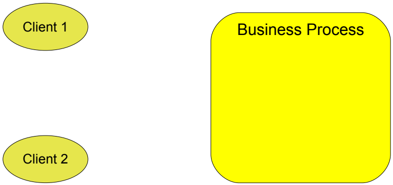
The message content permits identifying the proper target instance

Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)
- Message correlation permits
  - integrating *asynchronous* services,
    that take from a few minutes to some days to complete
  - tieing messages together in order to build long-lived interactions
  - implementing statefull *multiparty* conversations

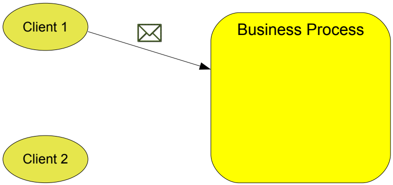
Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)
- Message correlation permits
  - integrating *asynchronous* services,
    that take from a few minutes to some days to complete
  - tieing messages together in order to build long-lived interactions
  - implementing statefull *multiparty* conversations



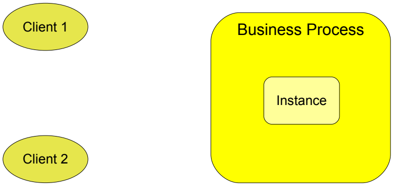
Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)
- Message correlation permits
  - integrating *asynchronous* services,
    that take from a few minutes to some days to complete
  - tieing messages together in order to build long-lived interactions
  - implementing statefull *multiparty* conversations



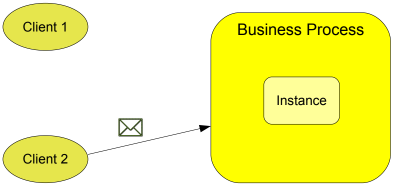
Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)
- Message correlation permits
  - integrating *asynchronous* services,
    that take from a few minutes to some days to complete
  - tieing messages together in order to build long-lived interactions
  - implementing statefull *multiparty* conversations



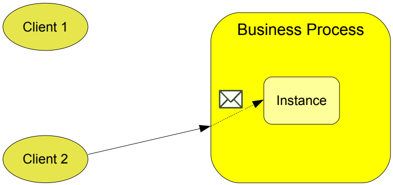
Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)
- Message correlation permits
  - integrating *asynchronous* services,
    that take from a few minutes to some days to complete
  - tieing messages together in order to build long-lived interactions
  - implementing statefull *multiparty* conversations



Instantiation & Message Correlation

- To serve clients' requests service instances are created
- When a message arrives, it must be delivered:
  - either to a new instance (new conversation)
  - or to an existing instance (old conversation)
- Message correlation permits
  - integrating *asynchronous* services,
    that take from a few minutes to some days to complete
  - tieing messages together in order to build long-lived interactions
  - implementing statefull *multiparty* conversations



Web Services Composition Languages

- Different organizations have been involved and are presently working on the design of languages for specifying business processes

- Two WS-BPEL's forerunners are
  - Microsoft's XLANG
    a block-structured language with basic control flow structures
    - e.g. sequence, switch (conditional), while (looping), all (parallel) and pick (choice based on timing or external events)
  - IBM's WSFL (Web Services Flow Language)
    a language for specifying arbitrary directed acyclic graphs

- Afterwards, the two proposals have been combined into a new language, WS-BPEL, that has been submitted to OASIS for standardization also by BEA systems, SAP and Siebel Systems

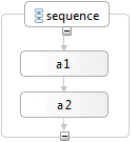
WS-BPEL

- Web Services Business Process Execution Language Version 2.0
- Is an OASIS standard (11 April 2007)
- Is the most widespread language for orchestration of Web Services
- Has an XML-based syntax and relies on the following XML-based specifications
  - WSDL for interfaces
  - XML Schema for types
  - XPath for expressions

WS-BPEL: basic activities

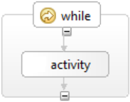
- empty to do nothing
- invoke to invoke an operation offered by a (partner) service
  - partner services are identified by *partner links*
- receive to wait for a request to arrive
- reply to send a message in reply to a previously received request
- assign to update the values of variables with new data

WS-BPEL: control flow activities



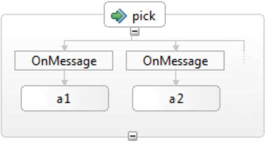
to perform a collection of activities in sequential order

to select exactly one activity for execution from two alternatives

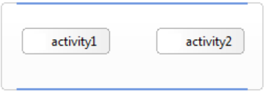


to repeat an activity as long as a given condition is true

WS-BPEL: control flow activities



to wait for one of several possible requests to arrive



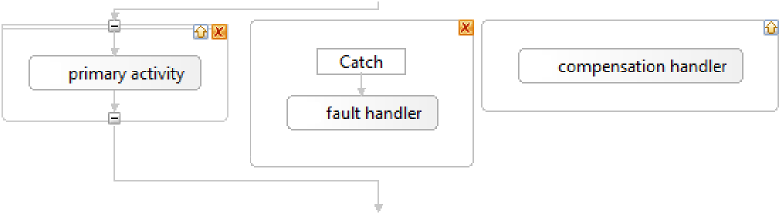
to concurrently perform a set of activities (flow activity)

WS-BPEL: fault and compensation

- *Fault handling*: similar to exception handling of ‘classic’ programming languages
- *Compensation*: execution of specific activities (attempting) to reverse the effects of previously executed activities

WS-BPEL: fault and compensation

- *Fault handling*: similar to exception handling of ‘classic’ programming languages
- *Compensation*: execution of specific activities (attempting) to reverse the effects of previously executed activities
- *Scope activity*: groups a primary activity together with fault handling activities and a compensation handling activity



WS-BPEL: fault and compensation

- exit to immediately terminate an instance
- throw to generate a fault from inside an instance
- rethrow to rethrow the fault that was originally caught by the immediately enclosing fault handler
- compensate to start compensation on all inner scopes that have already completed successfully, in the *reverse order* of completion
- compensateScope to start compensation of a specified inner scope that has already completed successfully

WS-BPEL: other aspects

- Termination and event handlers within scope activities
- Synchronization dependencies within flow activities
- repeatUntil and forEach activities
- Timed activities

WS-BPEL engines

- Three of the most known freely available WS-BPEL engines

Oracle BPEL Process Manager 10.1.3

`http://www.oracle.com/technology/bpel`

ActiveBPEL 4.1

`http://www.activevos.com`

Apache ODE 1.1.1

`http://ode.apache.org`

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. $\mathtt{web}\pi$, $\mathtt{web}\pi_\infty$, CSP/π-calculus + transactions, ...

Process Calculi for SOC: an overview

Process calculi for SOC can be classified according to the approach 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*

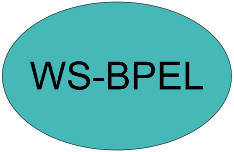
- **Sessions**: the link is determined by a private channel that is implicitly created when the first message exchange of a conversation takes place
- **Correlations**: the link is determined by correlation values included in the exchanged messages
  - *stateful*: every service instance has an explicit 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

A gentle introduction to COWS

COWS: a Calculus for Orchestration of Web Services

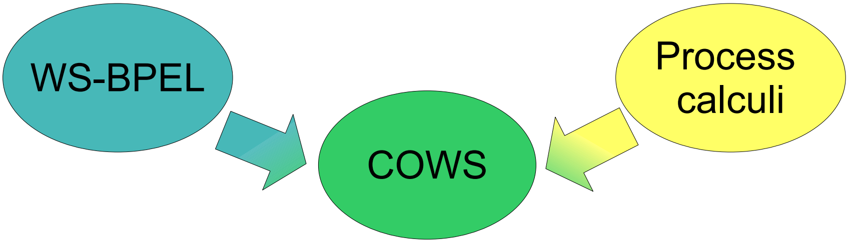


- Inspired by
  - the OASIS standard WS-BPEL for WS orchestration
  - previous work on process calculi
- Indeed, COWS intends to be a foundational model not specifically tight to Web services' current technologies
- COWS combines in an original way a number of constructs and features borrowed from well-known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

COWS: a Calculus for Orchestration of Web Services

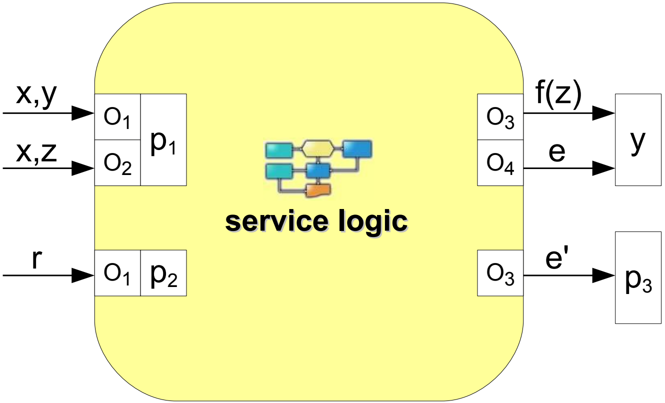
- Inspired by
  - the OASIS standard WS-BPEL for WS orchestration
  - previous work on process calculi
- Indeed, COWS intends to be a foundational model not specifically tight to Web services' current technologies
- COWS combines in an original way a number of constructs and features borrowed from well-known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

COWS: a Calculus for Orchestration of Web Services

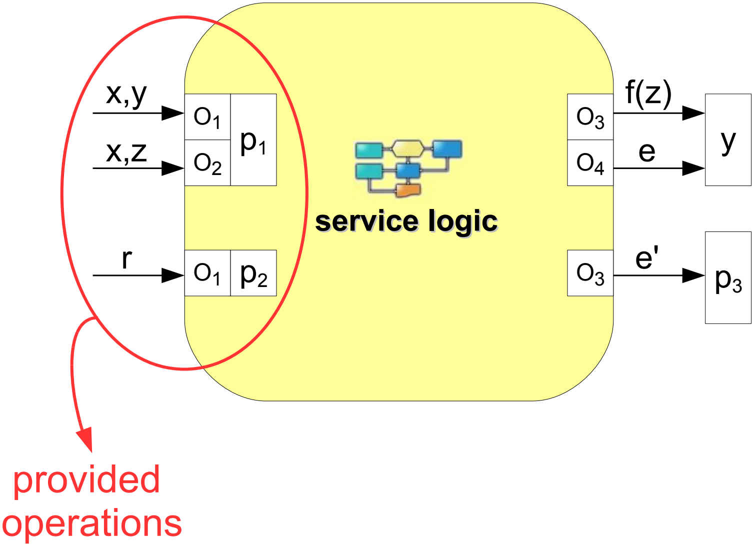


- Inspired by
  - the OASIS standard WS-BPEL for WS orchestration
  - previous work on process calculi
- Indeed, COWS intends to be a foundational model not specifically tight to Web services' current technologies
- COWS combines in an original way a number of constructs and features borrowed from well-known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

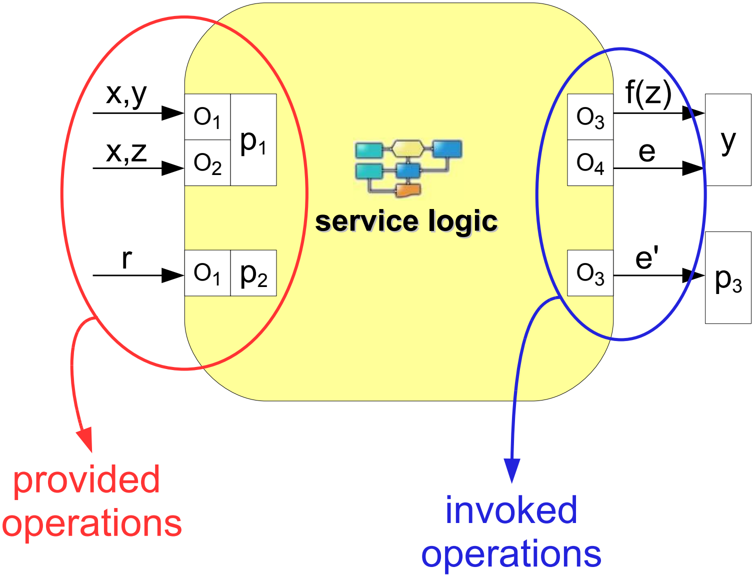
The notion of service in COWS



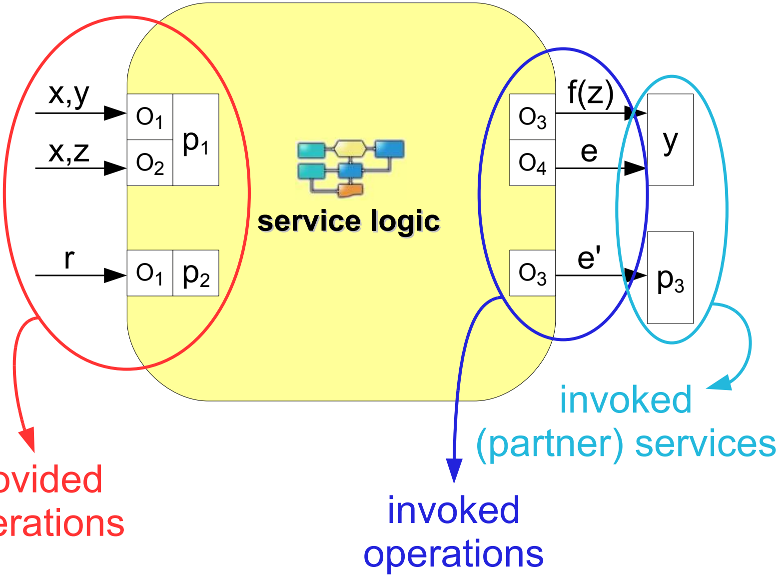
The notion of service in COWS



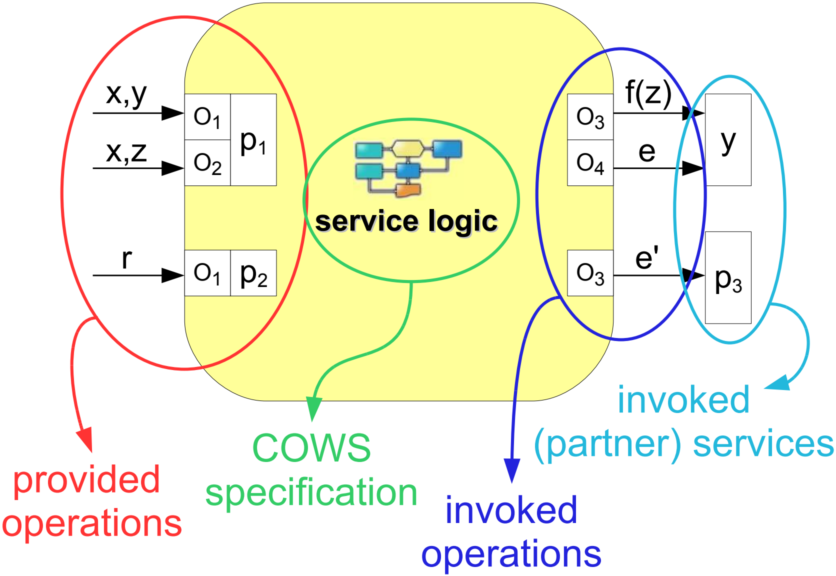
The notion of service in COWS



The notion of service in COWS



The notion of service in COWS



COWS in 3 steps

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

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

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

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

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

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

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

$$
\left.\bullet\ \text{pattern} - \text{matching}\right\} \text{Klaim}
$$

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
- $\bar{\ }$ 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* $\mid$ *names*
w: *variables* $\mid$ *values*

Choice

- $+$ abbreviates binary choice, while empty choice will be denoted by $\mathbf{0}$

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

$s ::=$		(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*

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)
$\mid$	$\sum_{i=0}^{r} g_i.s_i$	(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*

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 $x \mapsto v$)

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 $x \mapsto v$)

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 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 operational semantics

Labelled transition rules

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

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

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

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

Matching function

$$\mathcal{M}(x, v) = \{x \mapsto v\}$$

$$\mathcal{M}(v, v) = \emptyset$$

$$\mathcal{M}(\langle\rangle, \langle\rangle) = \emptyset$$

$$\frac{\mathcal{M}(w_1, v_1) = \sigma_1 \quad \mathcal{M}(\bar{w}_2, \bar{v}_2) = \sigma_2}{\mathcal{M}((w_1, \bar{w}_2), (v_1, \bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

μCOWS^m operational semantics

Labelled transition rules

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

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

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

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)

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

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

$$\mathcal{M}(x, v) = \{x \mapsto v\} \qquad \mathcal{M}(v, v) = \emptyset \qquad \mathcal{M}(\langle\rangle, \langle\rangle) = \emptyset$$

$$\frac{\mathcal{M}(w_1, v_1) = \sigma_1 \quad \mathcal{M}(\bar{w}_2, \bar{v}_2) = \sigma_2}{\mathcal{M}((w_1, \bar{w}_2), (v_1, \bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

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

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

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

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

μCOWS^m: 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

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

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

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

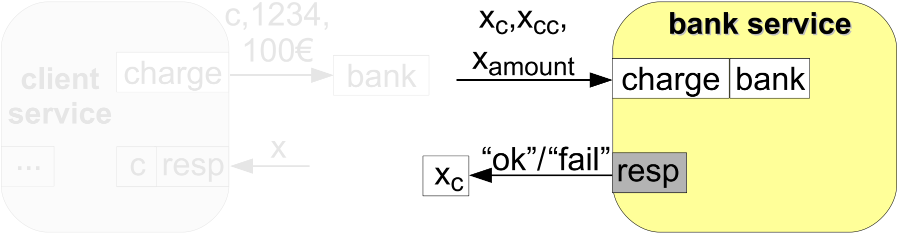
$$\frac{s_1 \xrightarrow{\alpha} s_1'}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}$$

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

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

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

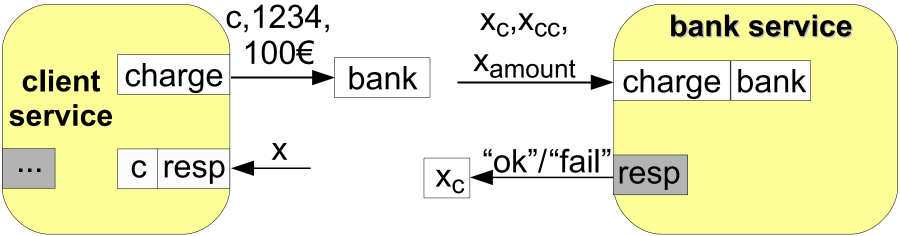
μCOWS^m: *simple* bank service example



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

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

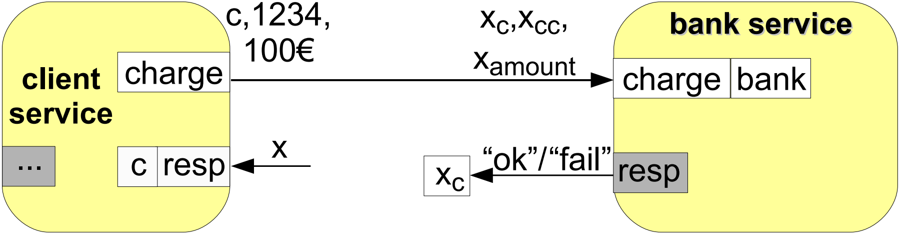
μCOWS^m: *simple* bank service example



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

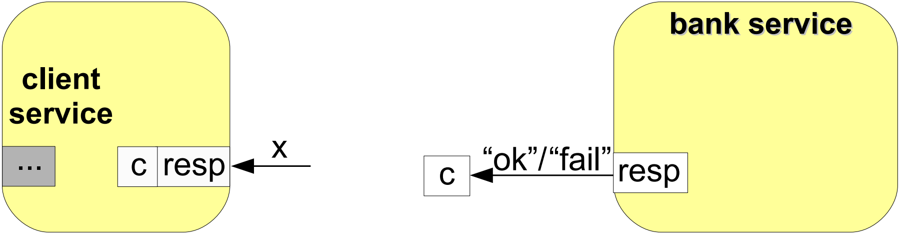
$[\mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}]$
$\mathsf{bank} \bullet \mathsf{charge}?\langle \mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}} \rangle.$
$\mathsf{x_c} \bullet \mathsf{resp}!\langle chk(x_{cc}, x_{amount}) \rangle$

μCOWS^m: *simple* bank service example



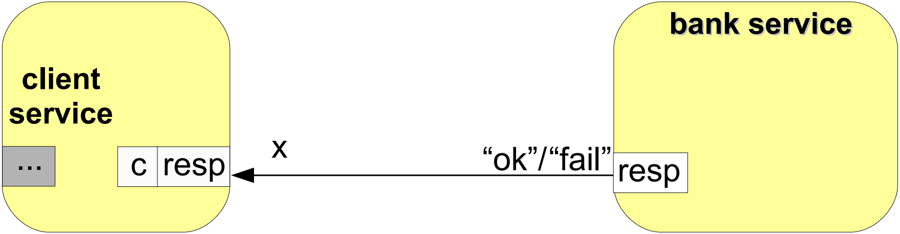
$$
\begin{array}{lll}
\mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, 1234, 100€ \rangle & & [\mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}] \\
\mid\ [\mathsf{x}]\,(\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x} \rangle.\mathsf{s} \mid \mathsf{s}') & \mid & \mathsf{bank} \bullet \mathsf{charge}?\langle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

μCOWS^m: *simple* bank service example



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

μCOWS^m: *simple* bank service example

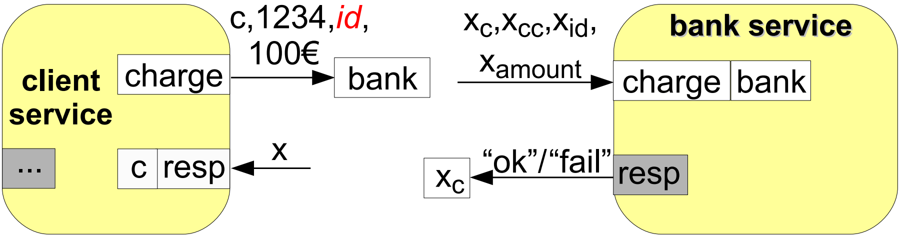


$$[x]\,(c \bullet resp?\langle x\rangle.s \mid s') \qquad\qquad \mid \qquad\qquad c \bullet resp!\langle \mathit{chk}(1234, 100€)\rangle$$

μCOWS^m: *simple* bank service example

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

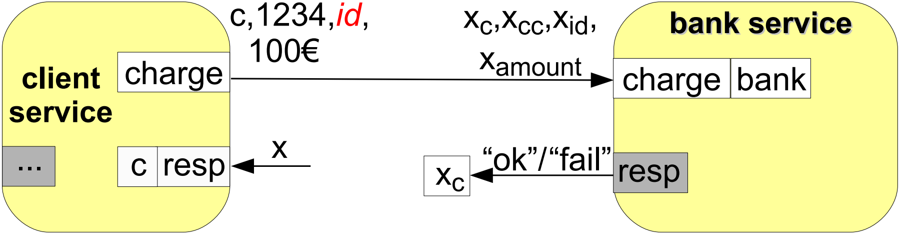
μCOWS^m: communication of private names



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

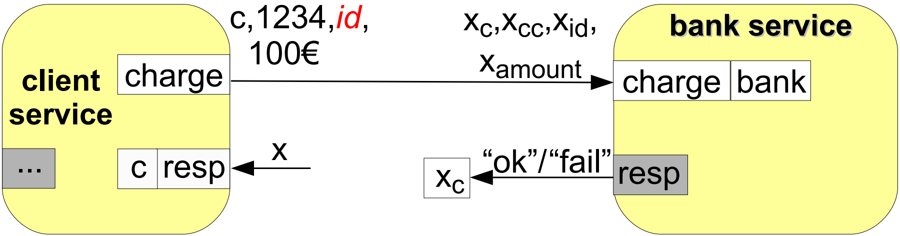
$[x_c, x_{cc}, x_{id}, x_{amount}]$
$\text{bank} \bullet \text{charge}?\langle x_c, x_{cc}, x_{id}, x_{amount}\rangle.$
$x_c \bullet \text{resp}!\langle \mathit{chk}(x_{cc}, x_{id}, x_{amount})\rangle$

μCOWS^m: communication of private names



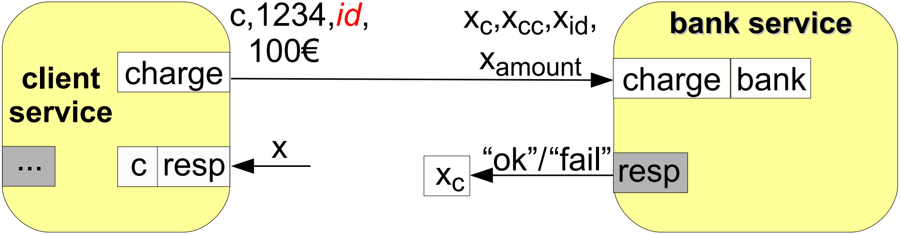
$$\color{red}{[\mathrm{id}]}\ \color{red}{(}\mathrm{bank} \bullet \mathrm{charge}!\langle \mathrm{c}, 1234, \color{red}{\mathrm{id}}, 100€\rangle \ \mid \ [\mathrm{x}]\,(\mathrm{c} \bullet \mathrm{resp}?\langle \mathrm{x}\rangle.\mathrm{s} \mid \mathrm{s}')\ \color{red}{)} \quad \mid \quad [\mathrm{x_c},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

μCOWS^m: communication of private names



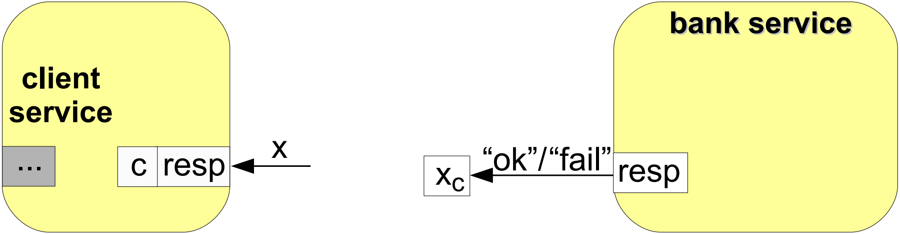
$\equiv$

μCOWS^m: communication of private names



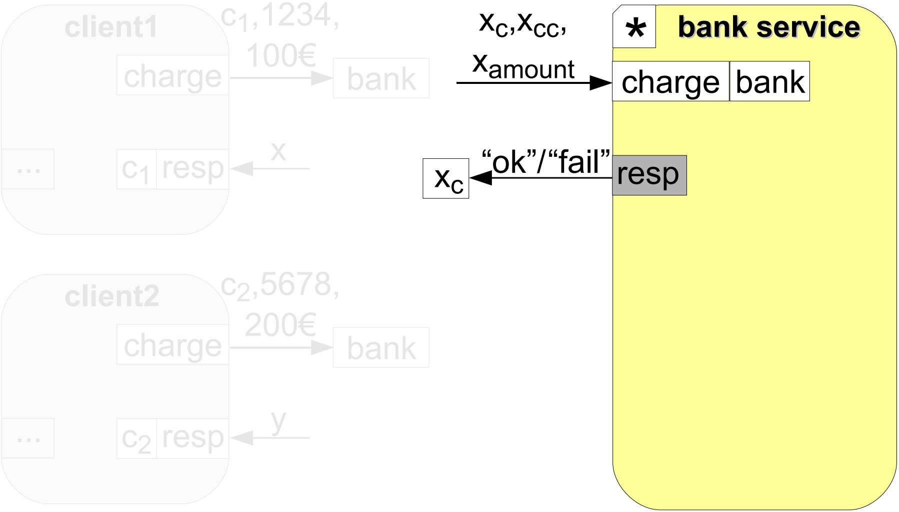
$$[\mathrm{id}, x_c, x_{cc}, x_{id}, x_{amount}] \left(\begin{pmatrix} \mathrm{bank} \bullet \mathrm{charge}!\langle \mathrm{c}, 1234, \mathrm{id}, 100€ \rangle \\ \mid [x]\,(\mathrm{c} \bullet \mathrm{resp}?\langle x \rangle.s \mid s') \end{pmatrix} \mid \begin{pmatrix} \mathrm{bank} \bullet \mathrm{charge}?\langle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

μCOWS^m: communication of private names



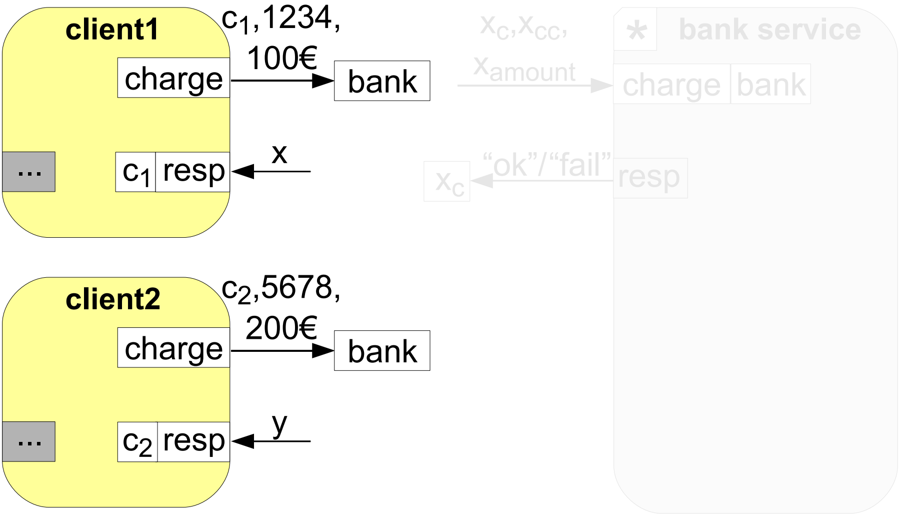
[id]
([x] (c • resp?⟨x⟩.s | s′) | c • resp!⟨*chk*(1234, id, 100€)⟩)

μCOWS^m: *persistent* bank service example



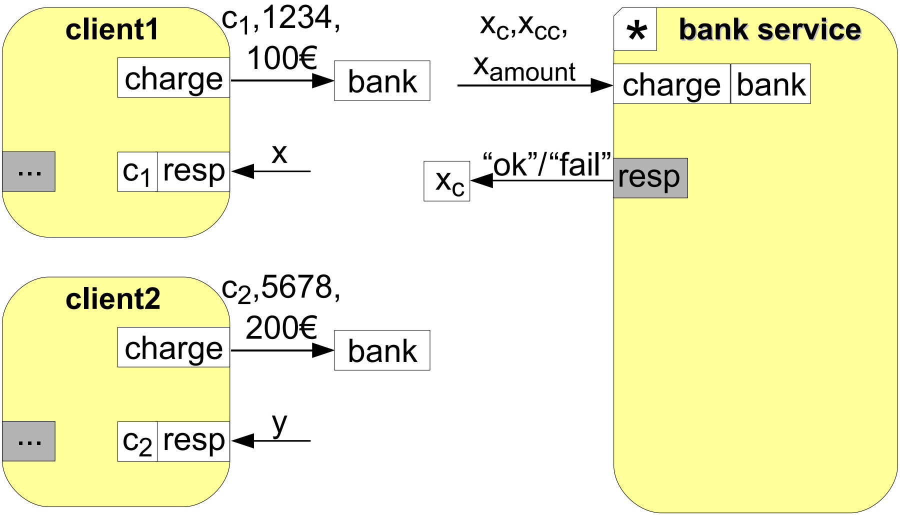
$$* \, [x_c, x_{cc}, x_{amount}] \, \mathrm{bank} \bullet \mathrm{charge}?\langle x_c, x_{cc}, x_{amount}\rangle . x_c \bullet \mathrm{resp}!\langle \mathit{chk}(x_{cc}, x_{amount})\rangle$$

μCOWS^m: *persistent* bank service example

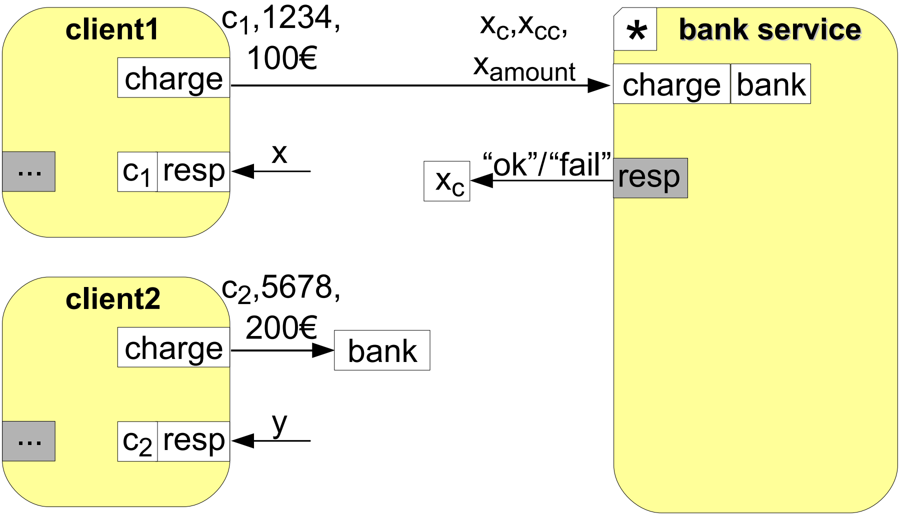


$$\mathrm{bank} \bullet \mathrm{charge}!\langle c_1, 1234, 100€\rangle \mid [x]\, c_1 \bullet \mathrm{resp}?\langle x\rangle.s_1$$
$$\mid\ \mathrm{bank} \bullet \mathrm{charge}!\langle c_2, 5678, 200€\rangle \mid [y]\, c_2 \bullet \mathrm{resp}?\langle y\rangle.s_2$$

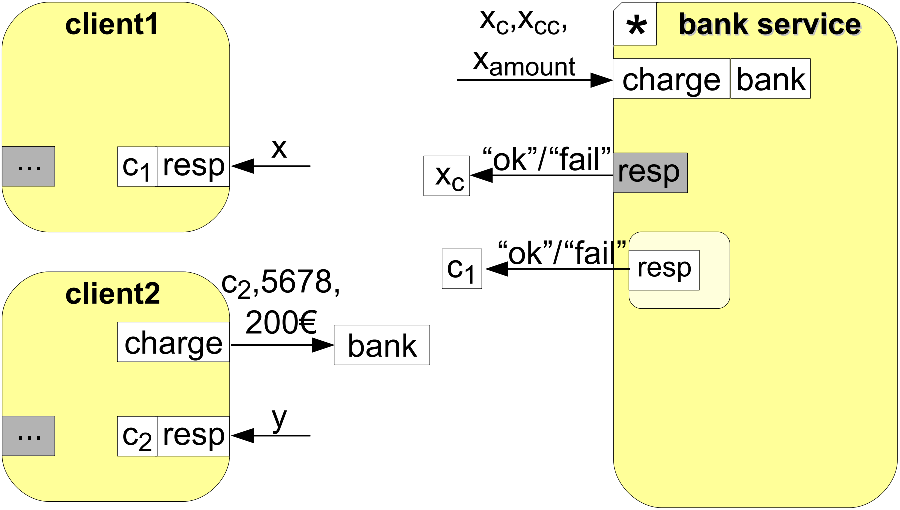
μCOWS^m: *persistent* bank service example



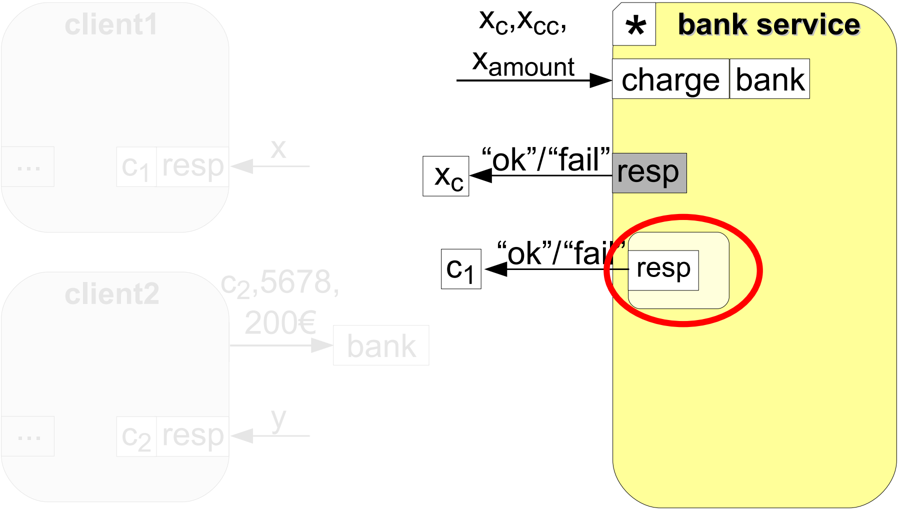
μCOWS^m: *persistent* bank service example



μCOWS^m: *persistent* bank service example



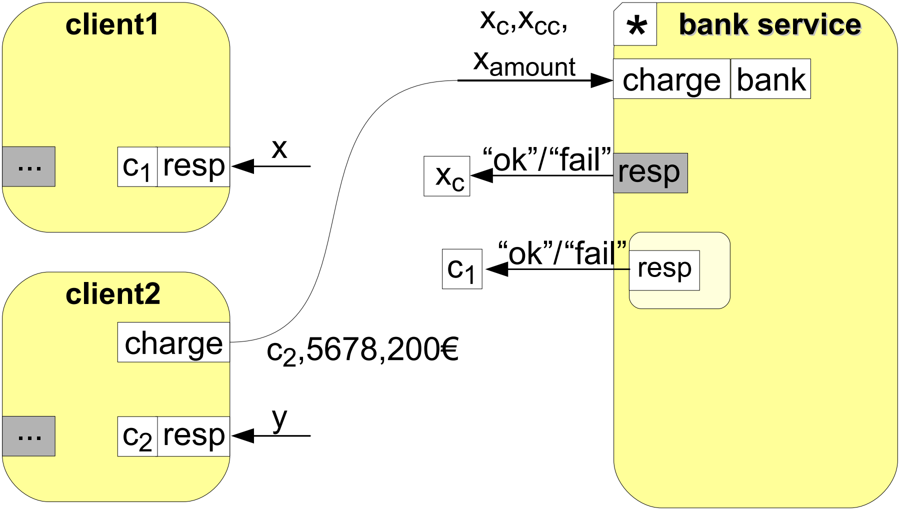
μCOWS^m: *persistent* bank service example



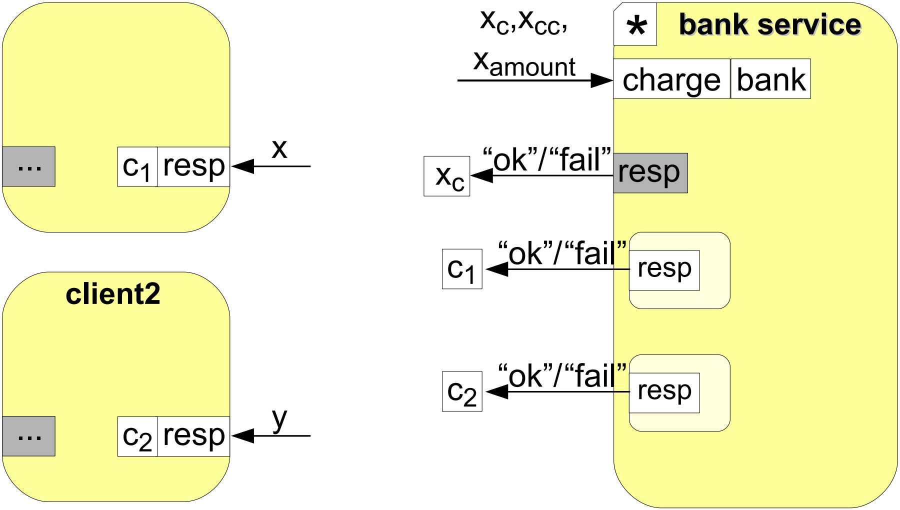
$* \, [x_c, x_{cc}, x_{amount}] \, \mathsf{bank} \bullet \mathsf{charge}?\langle x_c, x_{cc}, x_{amount} \rangle . x_c \bullet \mathsf{resp}!\langle \mathit{chk}(x_{cc}, x_{amount}) \rangle$

$| \;\; \mathsf{c}_1 \bullet \mathsf{resp}!\langle \mathit{chk}(1234, 100€) \rangle$

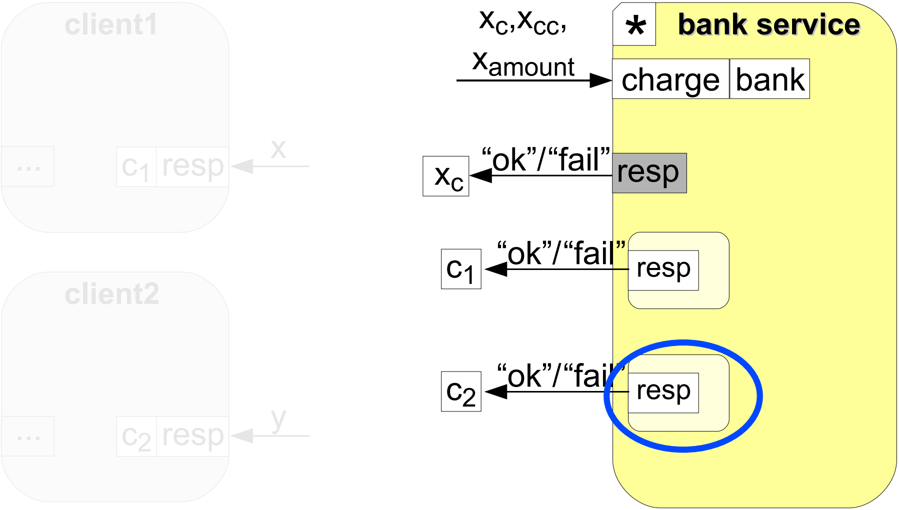
μCOWS^m: *persistent* bank service example



μCOWS^m: *persistent* bank service example

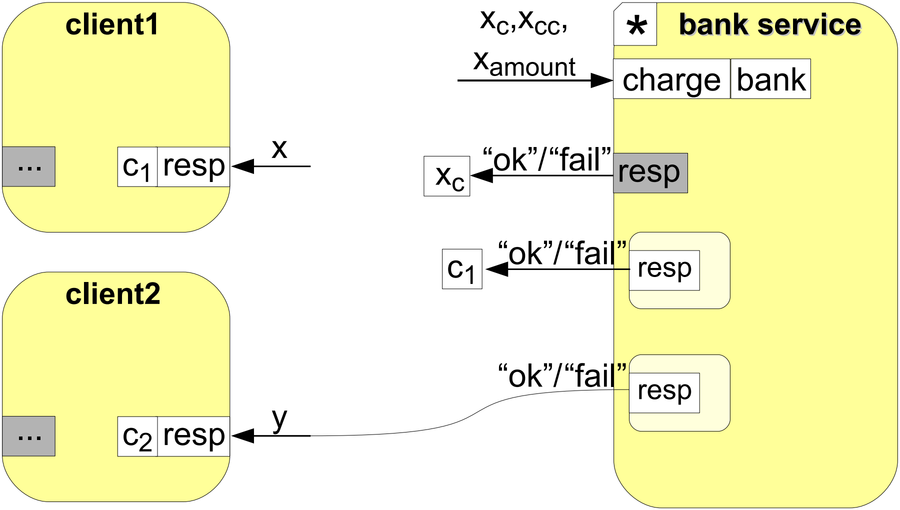


μCOWS^m: *persistent* bank service example

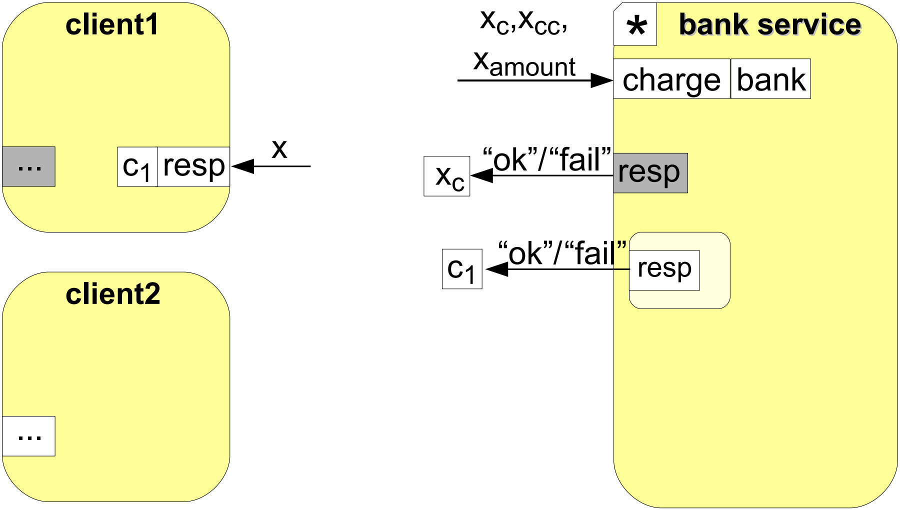


$*\,[x_c, x_{cc}, x_{amount}]\,\mathrm{bank} \bullet \mathrm{charge}?\langle x_c, x_{cc}, x_{amount}\rangle . x_c \bullet \mathrm{resp}!\langle \mathit{chk}(x_{cc}, x_{amount})\rangle$
$\mid\; c_1 \bullet \mathrm{resp}!\langle \mathit{chk}(1234, 100€)\rangle \;\mid\; c_2 \bullet \mathrm{resp}!\langle \mathit{chk}(5678, 200€)\rangle$

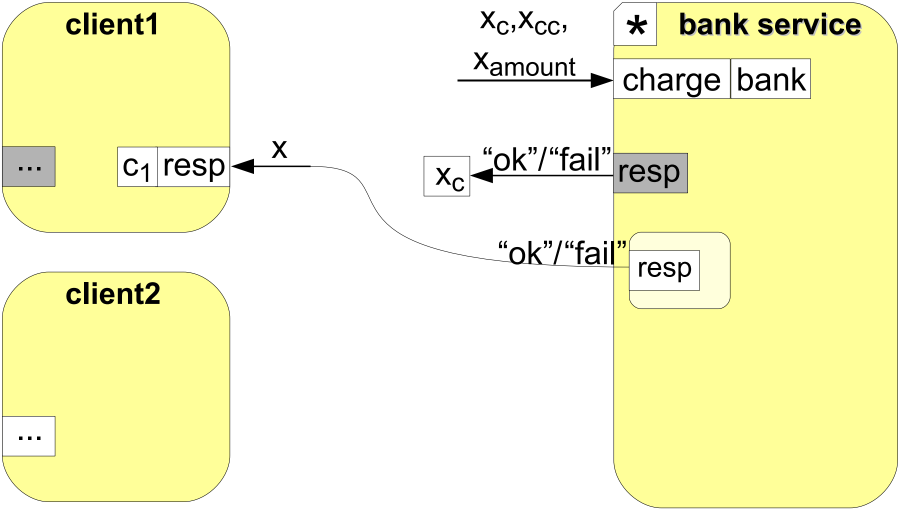
μCOWS^m: *persistent* bank service example



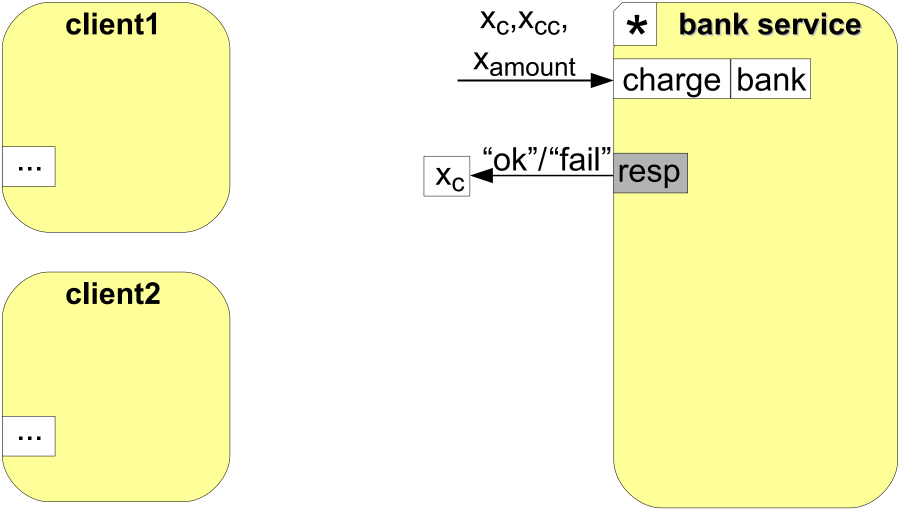
μCOWS^m: *persistent* bank service example



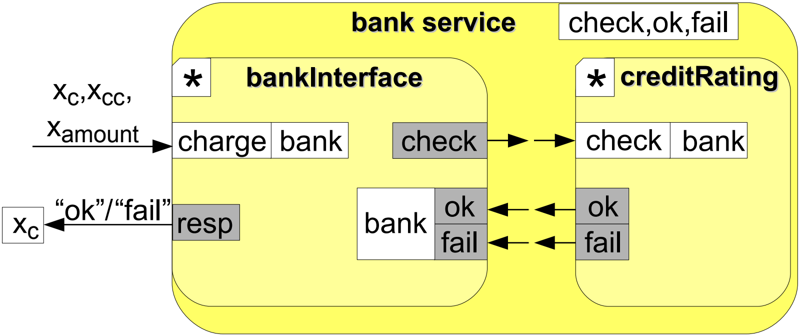
μCOWS^m: *persistent* bank service example



μCOWS^m: *persistent* bank service example

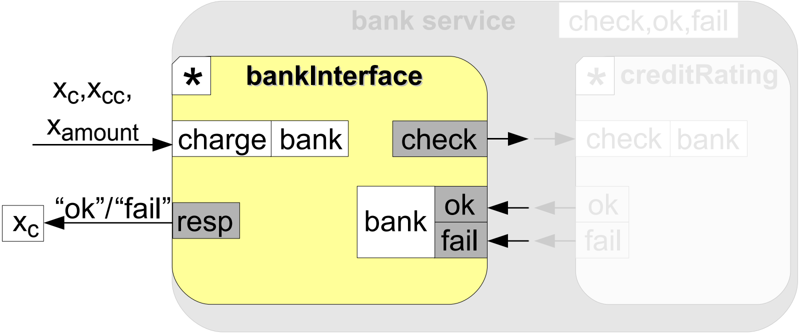


μCOWS^m: *compound* bank service example



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

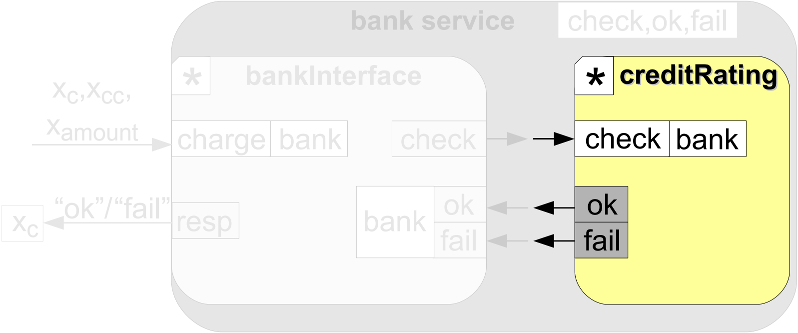
μCOWS^m: *compound* bank service example



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

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

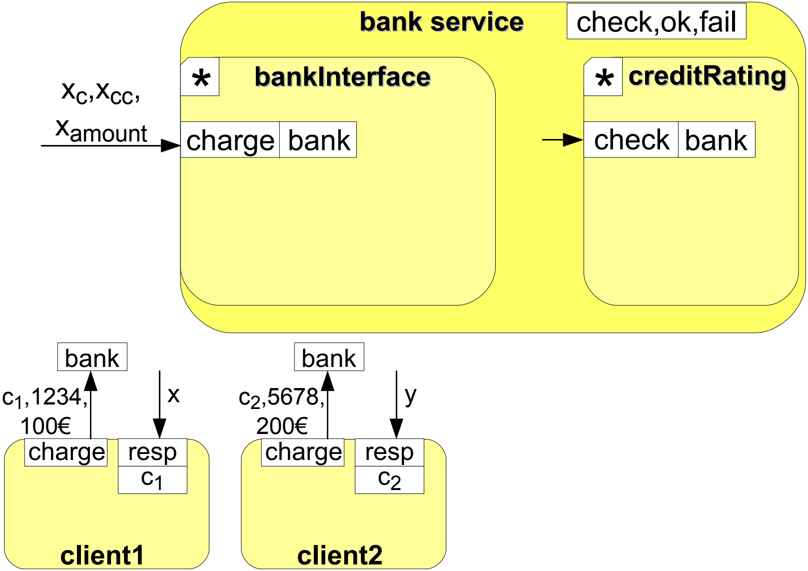
μCOWS^m: *compound* bank service example



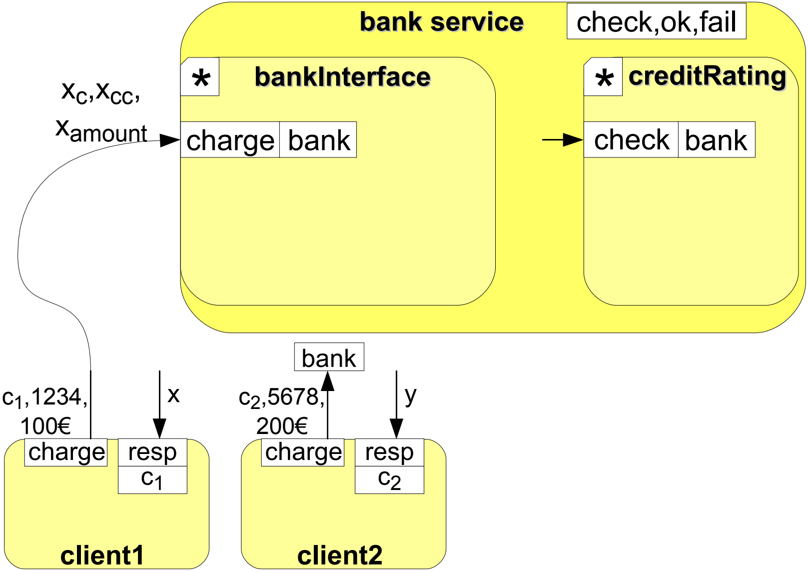
$$[\mathsf{check}, \mathsf{ok}, \mathsf{fail}]\,(\,*\,\mathsf{bankInterface} \mid *\,\mathsf{creditRating}\,)$$

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

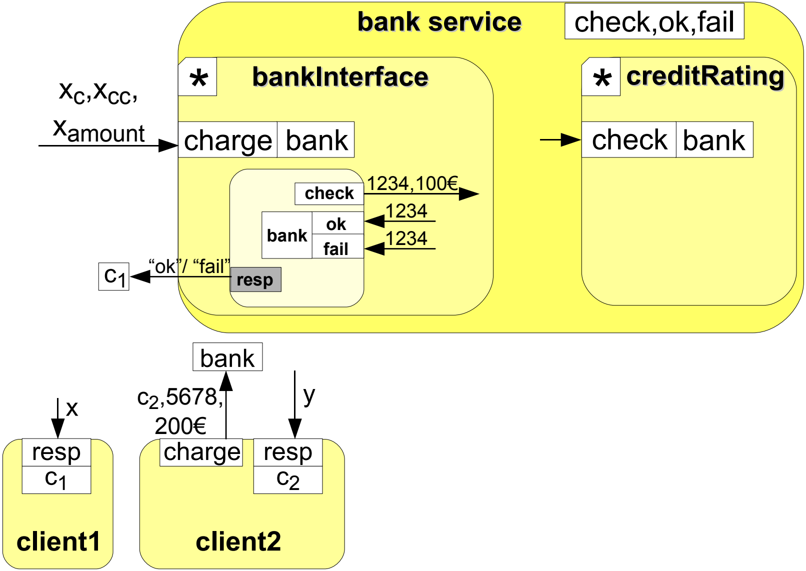
μCOWS^m: *compound* bank service example



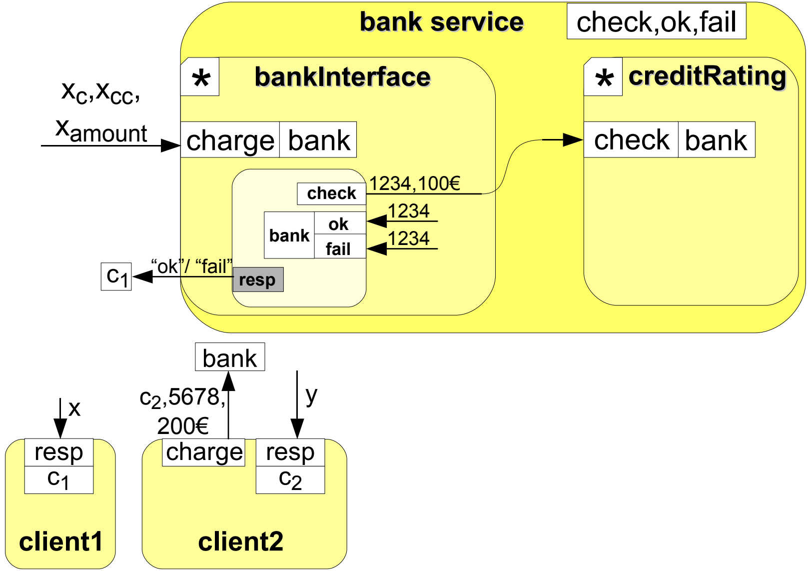
μCOWS^m: *compound* bank service example



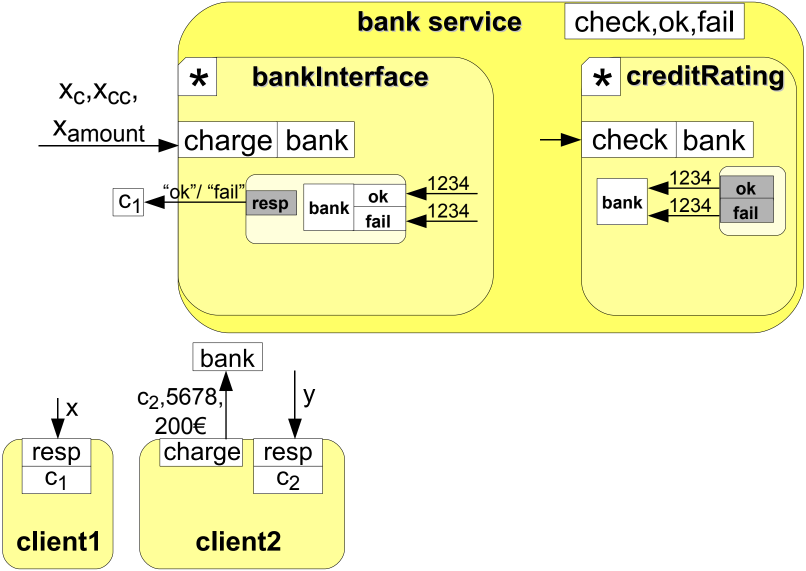
μCOWS^m: *compound* bank service example



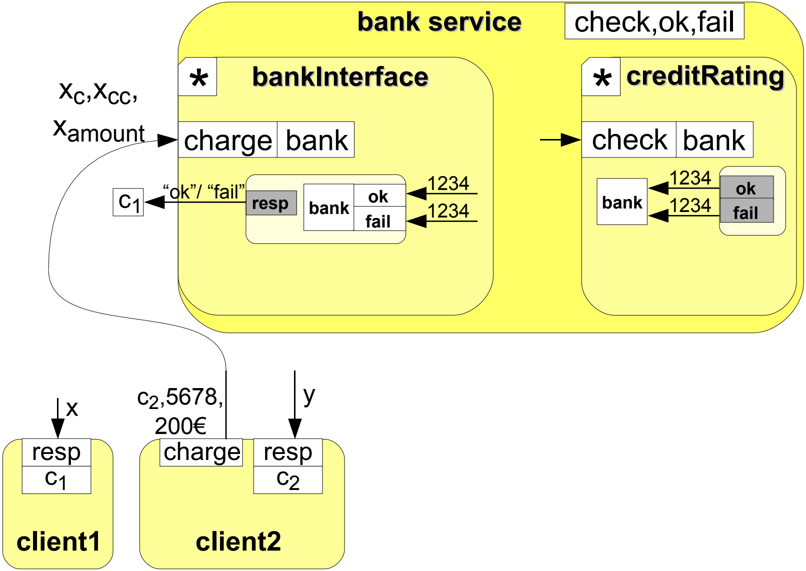
μCOWS^m: *compound* bank service example



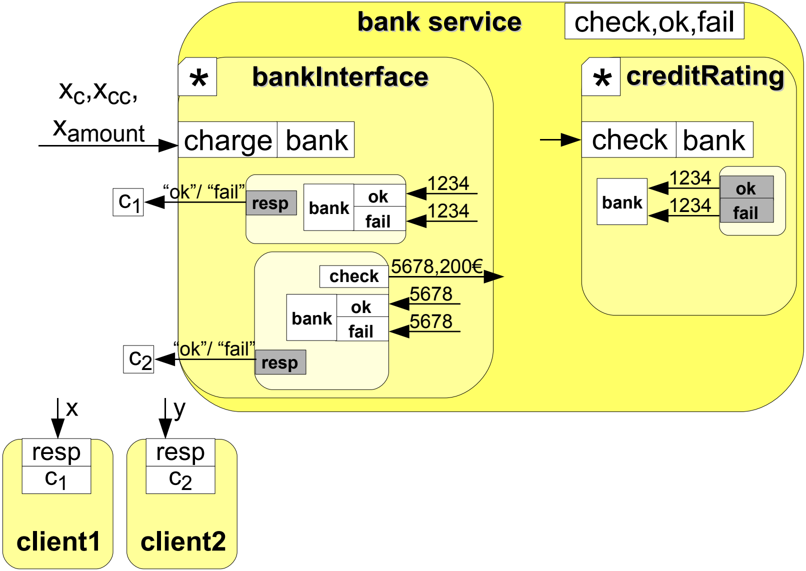
μCOWS^m: *compound* bank service example



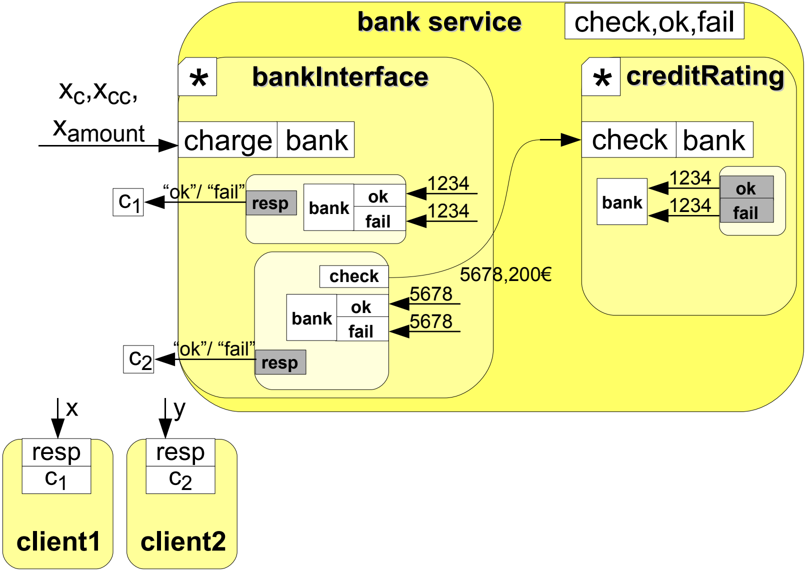
μCOWS^m: *compound* bank service example



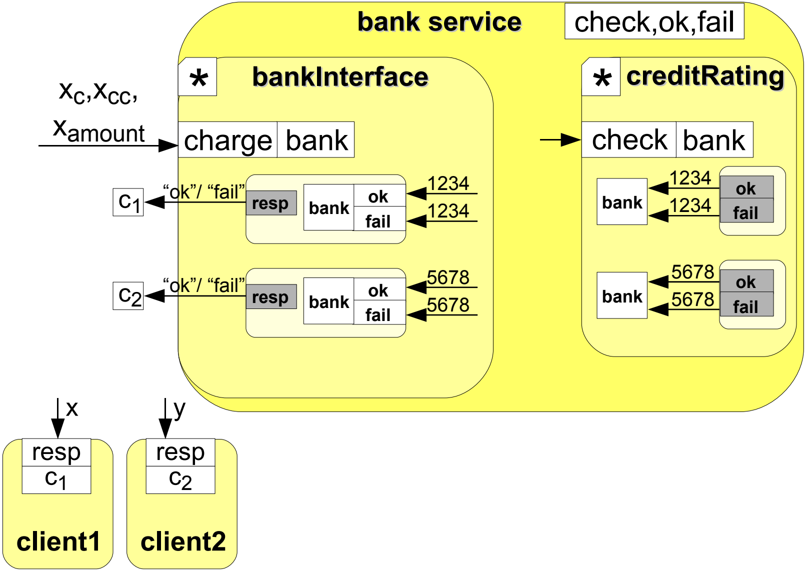
μCOWS^m: *compound* bank service example



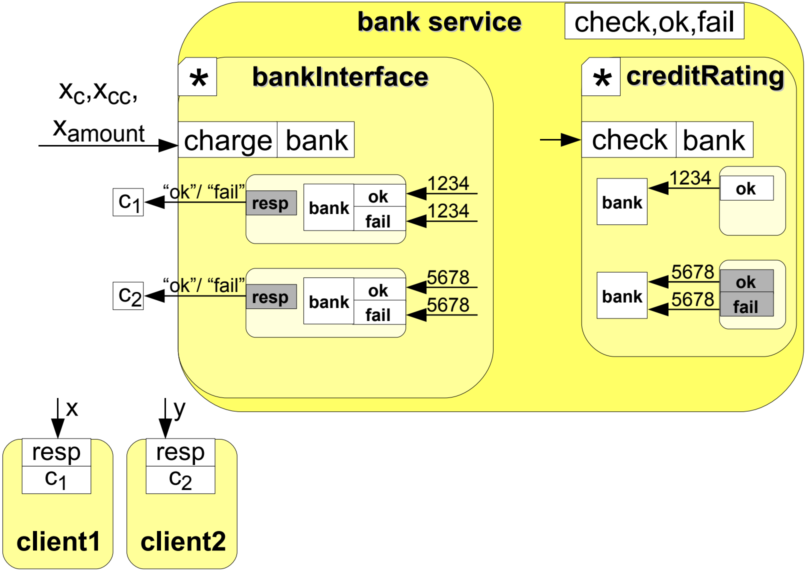
μCOWS^m: *compound* bank service example



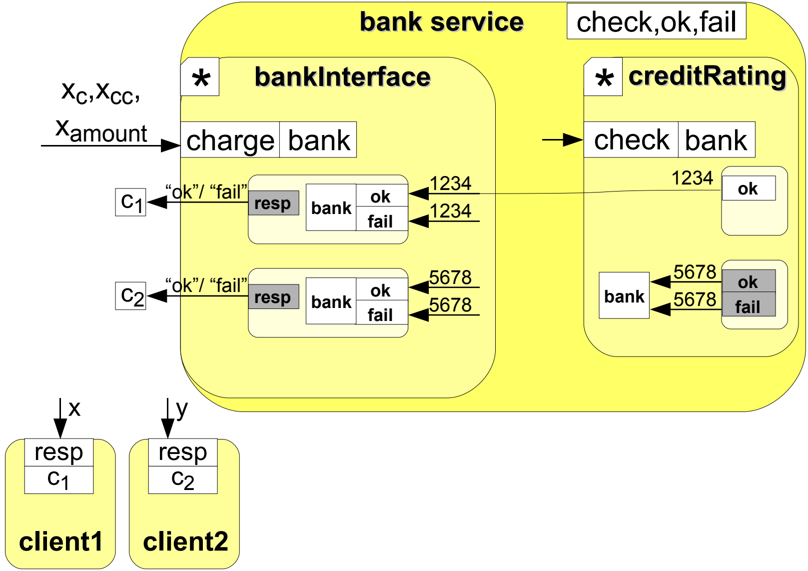
μCOWS^m: *compound* bank service example



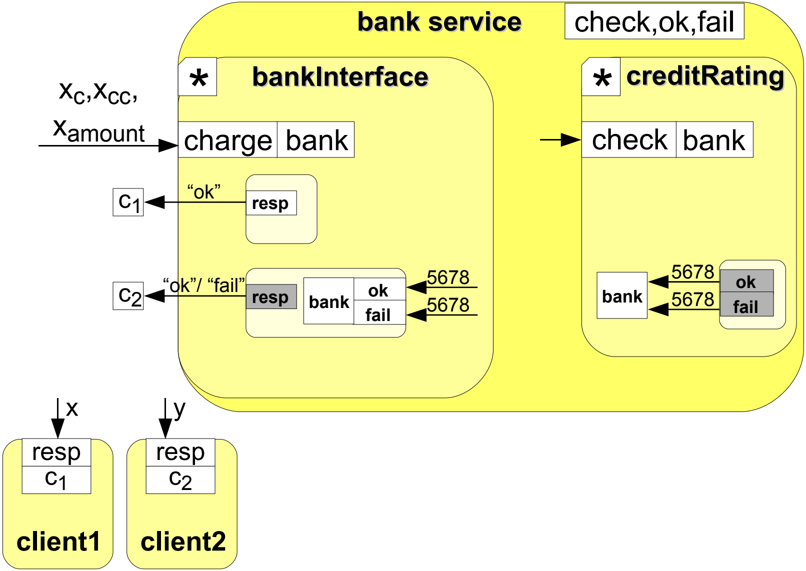
μCOWS^m: *compound* bank service example



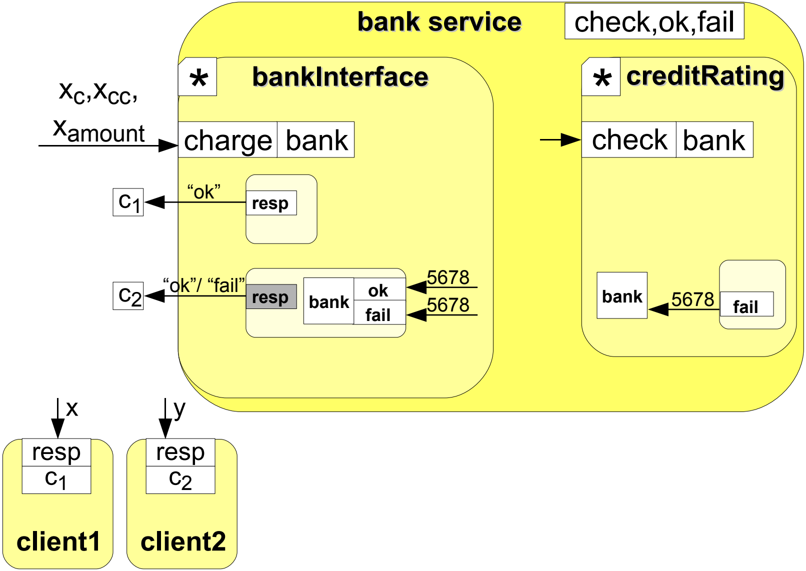
μCOWS^m: *compound* bank service example



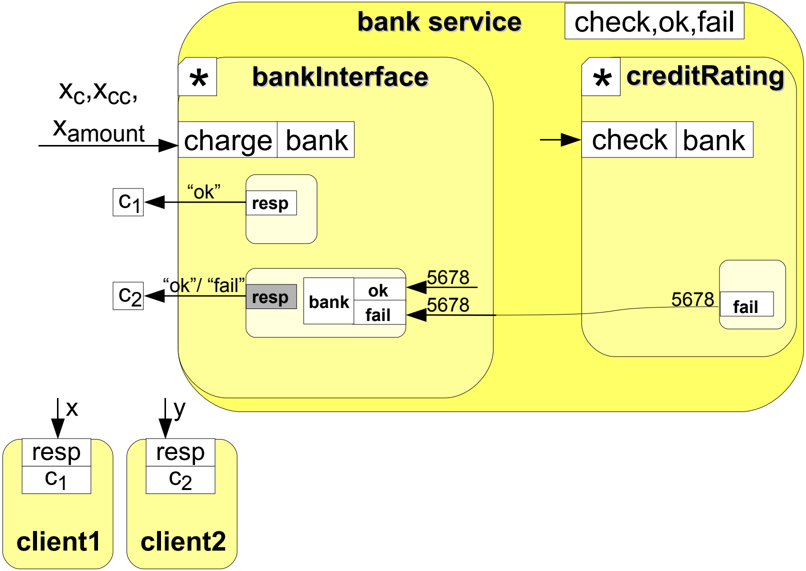
μCOWS^m: *compound* bank service example



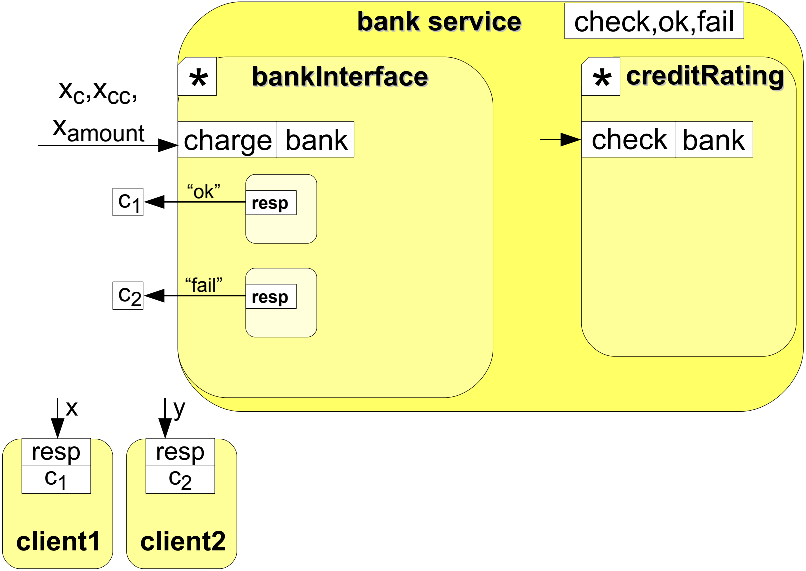
μCOWS^m: *compound* bank service example



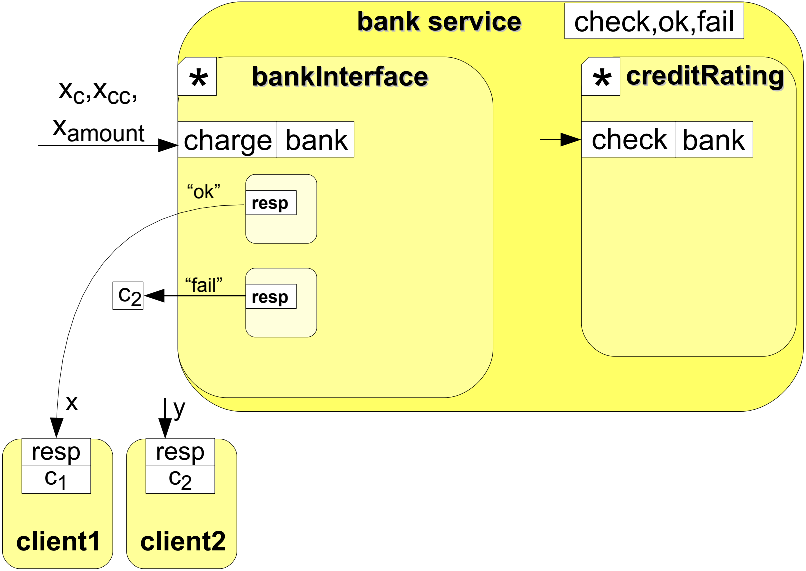
μCOWS^m: *compound* bank service example



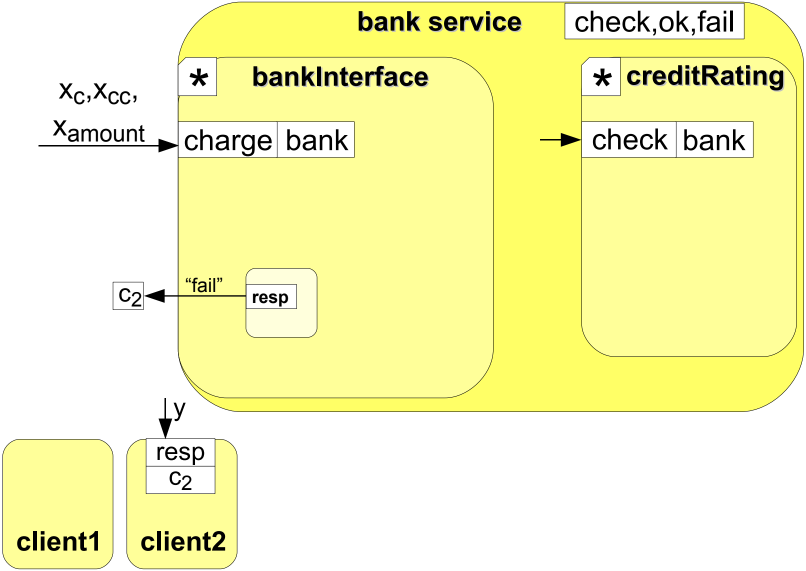
μCOWS^m: *compound* bank service example



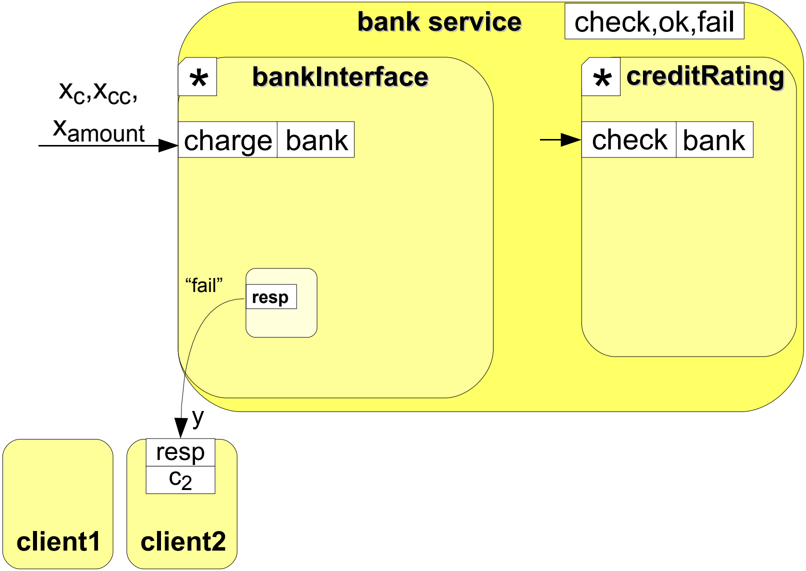
μCOWS^m: *compound* bank service example



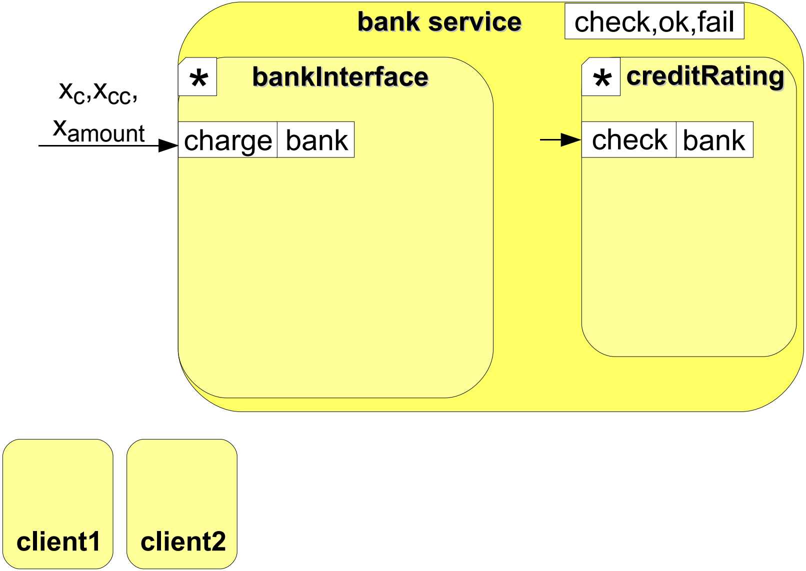
μCOWS^m: *compound* bank service example



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

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

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)

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

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

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

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 operational semantics

Labelled transition rules

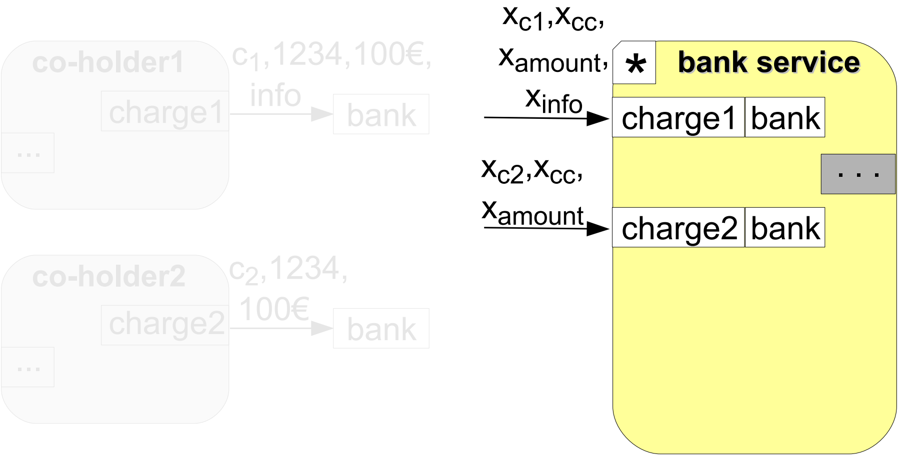
$$\frac{[\![\bar{\epsilon}]\!] = \bar{v}}{\mathrm{n}!\bar{\epsilon} \xrightarrow{\mathrm{n} \lhd \bar{v}} \mathbf{0}} \qquad \frac{1 \le j \le r}{\sum_{i=1}^{r} \mathrm{n}_i?\bar{w}_i.s_i \xrightarrow{\mathrm{n}_j \rhd \bar{w}_j} s_j} \qquad \frac{s \xrightarrow{\alpha} s' \quad u \notin \mathrm{u}(\alpha)}{[u]\,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

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

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

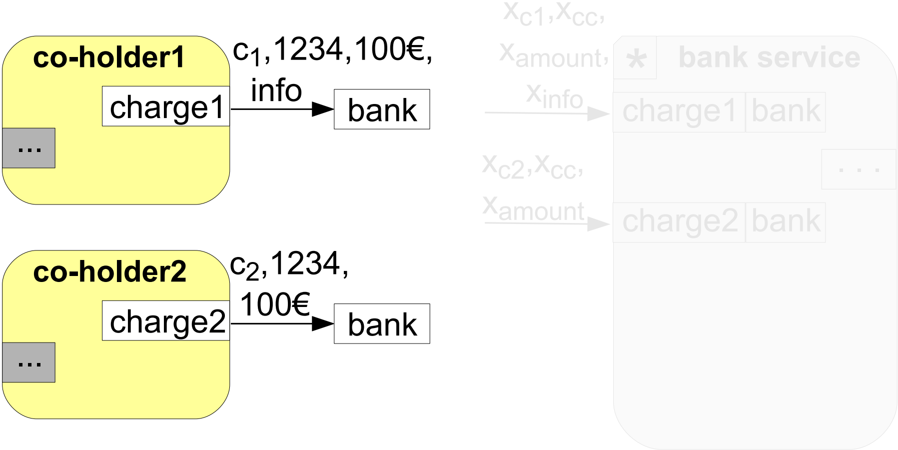
$$\frac{s \xrightarrow{\mathrm{n}\, \sigma \uplus \{x \mapsto v\}\, \ell\, \bar{v}} s'}{[x]\, s \xrightarrow{\mathrm{n}\, \sigma\, \ell\, \bar{v}} s' \cdot \{x \mapsto v\}}$$

μCOWS: *joint account* service example



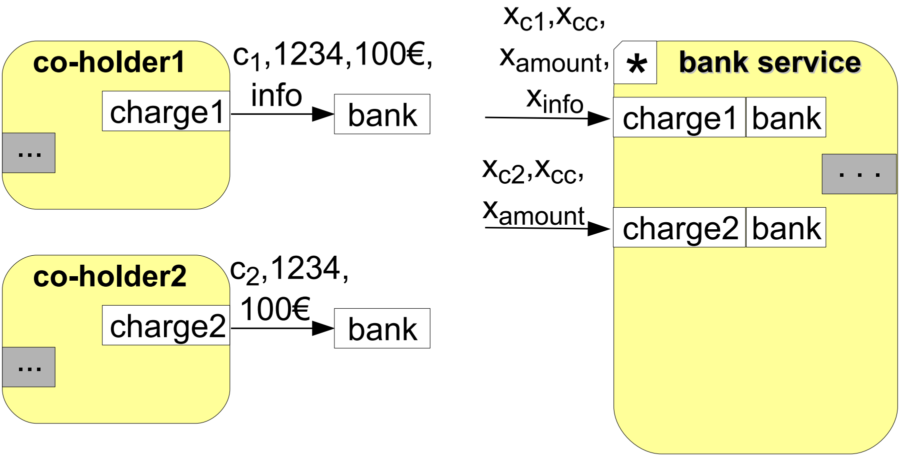
$$
\begin{aligned}
* \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (& \, \mathrm{bank} \bullet \mathrm{charge1} ? \langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
\mid & \, \mathrm{bank} \bullet \mathrm{charge2} ? \langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
\mid \, (&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

μCOWS: *joint account* service example



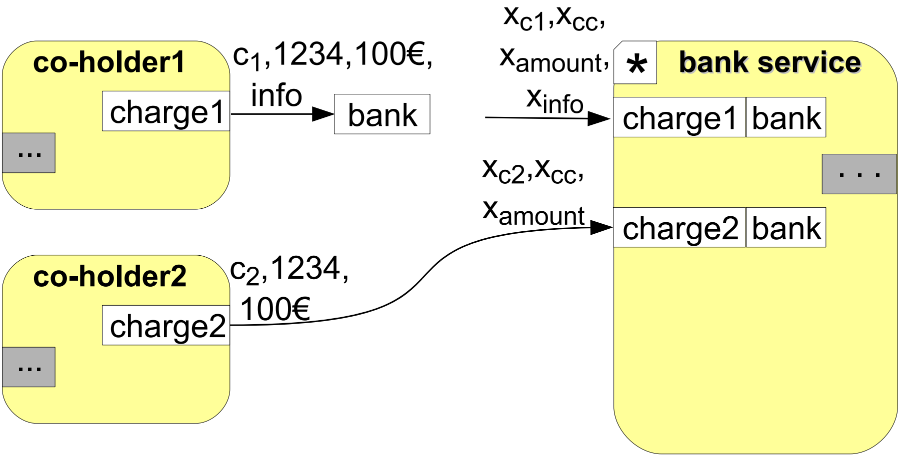
$$
\begin{aligned}
* \;\; [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}]\, (\, & \mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
\mid\; & \mathrm{bank} \bullet \mathrm{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
\mid\; (\, \mathrm{bank}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

μCOWS: *joint account* service example



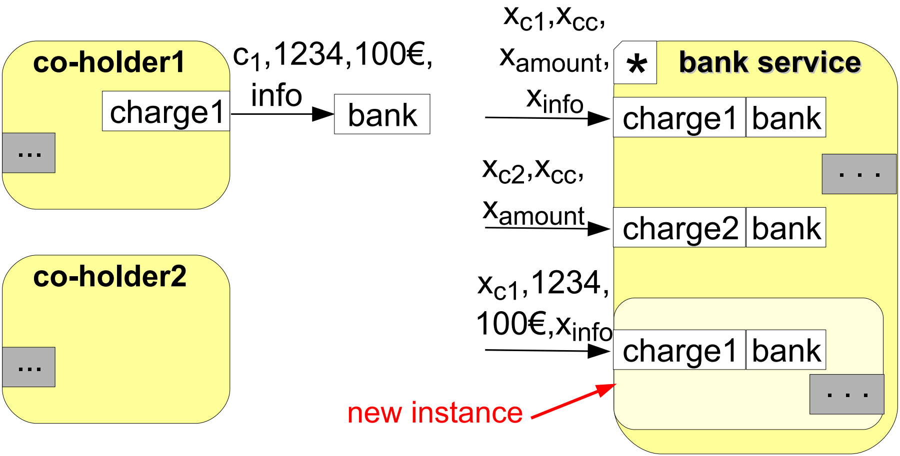
$$
\begin{array}{l}
* \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (\, bank \bullet charge1?\langle x_{c1}, x_{cc}, x_{amount}, x_{info}\rangle.s_1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \mid \; bank \bullet charge2?\langle x_{c2}, x_{cc}, x_{amount}\rangle.s_2 \,) \\
\mid \;(\, bank \bullet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

μCOWS: *joint account* service example



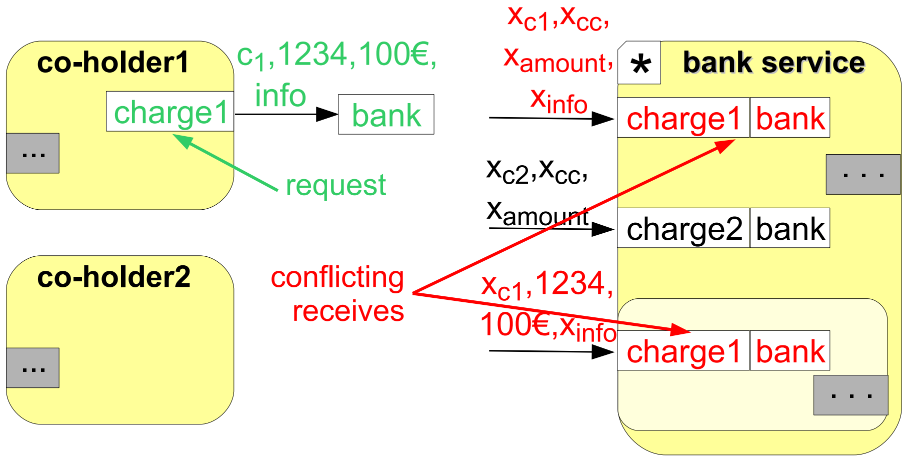
$$
\begin{aligned}
* \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (& \, \text{bank} \bullet \text{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
& \mid \text{bank} \bullet \text{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2 \,) \\
\mid \, (& \, \text{bank}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

μCOWS: *joint account* service example



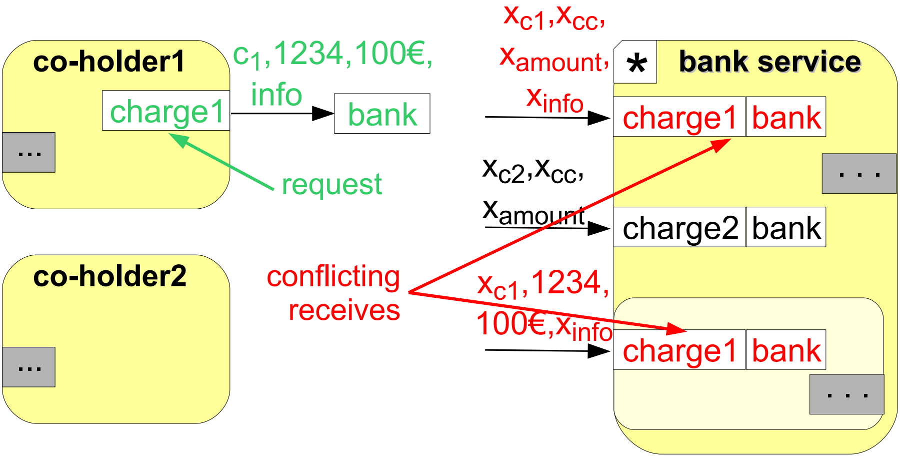
$$
\begin{array}{l}
*\,[x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}]\,(\,\mathrm{bank} \bullet \mathrm{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info}\rangle.s_1 \\
\qquad\qquad\qquad\qquad\qquad\quad \mid\; \mathrm{bank} \bullet \mathrm{charge2}?\langle x_{c2}, x_{cc}, x_{amount}\rangle.s_2\,)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

μCOWS: *joint account* service example



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

μCOWS: *joint account* service example

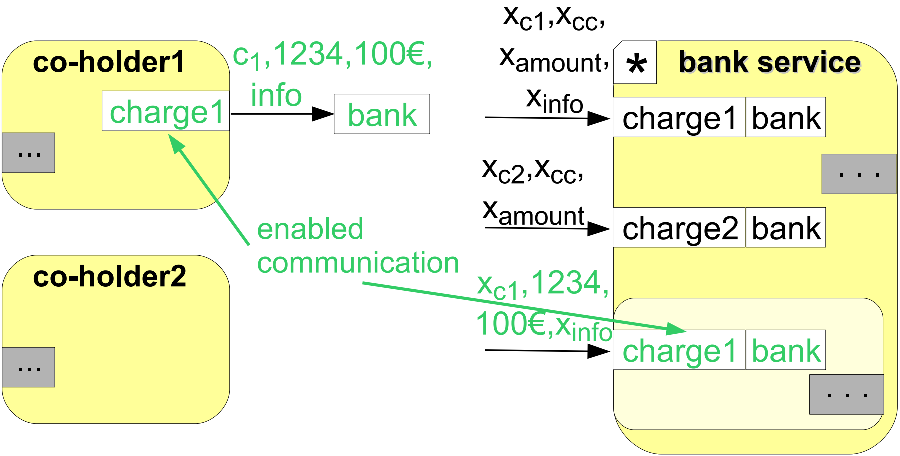


Multiple start activities

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

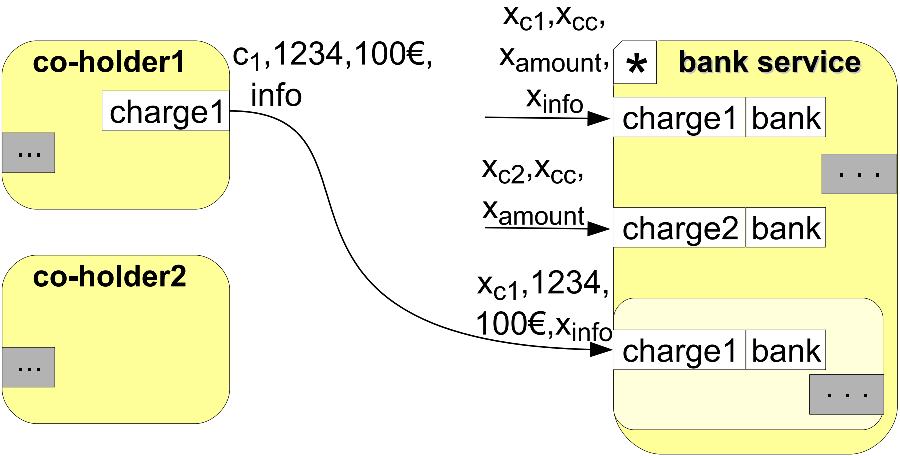
- the first incoming message triggers creation of a service instance
- subsequent messages are delivered to the created instance

μCOWS: *joint account* service example



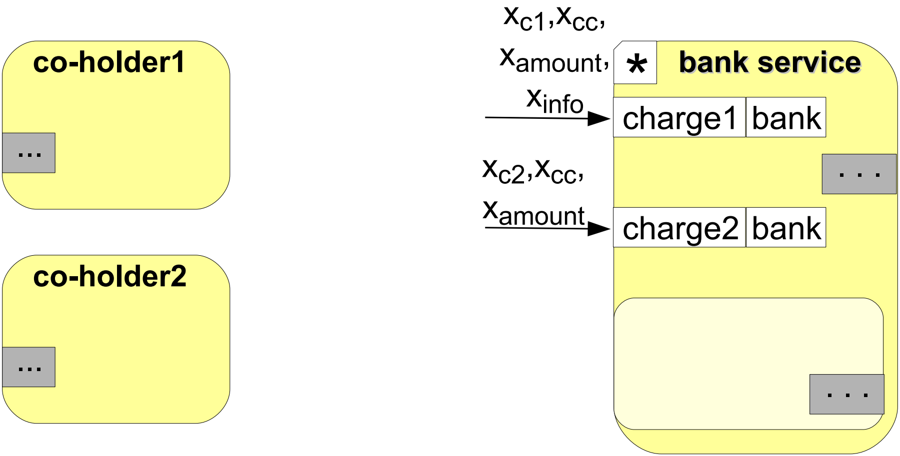
$$
\begin{aligned}
& * \, [x_{c1}, x_{c2}, x_{cc}, x_{amount}, x_{info}] \, (\, \text{bank} \bullet \text{charge1}?\langle x_{c1}, x_{cc}, x_{amount}, x_{info} \rangle . s_1 \\
& \qquad\qquad\qquad\qquad\qquad \mid \; \text{bank} \bullet \text{charge2}?\langle x_{c2}, x_{cc}, x_{amount} \rangle . s_2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

μCOWS: *joint account* service example



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

μCOWS: *joint account* service example



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

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

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

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

'Blind date' session joining

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

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

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

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

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

Parallel with priority as a coordination mechanism

‘Blind date’ session joining

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

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

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

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

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 *where* construct, that prunes threads selectively)

Syntax of COWS

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

(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

- $\{|\mathbf{0}|\} \;\equiv\; \mathbf{0} \qquad \{|\,\{|s|\}\,|\} \;\equiv\; \{|s|\} \qquad \{|[e]\,s|\} \;\equiv\; [e]\,\{|s|\}$
- $s_1 \mid [e]\,s_2 \;\equiv\; [e]\,(s_1 \mid s_2) \quad \text{if } e \notin \mathrm{fe}(s_1) \cup \mathrm{fk}(s_2)$
  - $\mathrm{fe}(s)$ 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 $\textbf{kill}(k)$ forces termination of all unprotected parallel activities inside an enclosing $[k]$, that stops the killing effect

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

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*

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

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

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 $\mathbf{kill}(k)$ 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

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

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

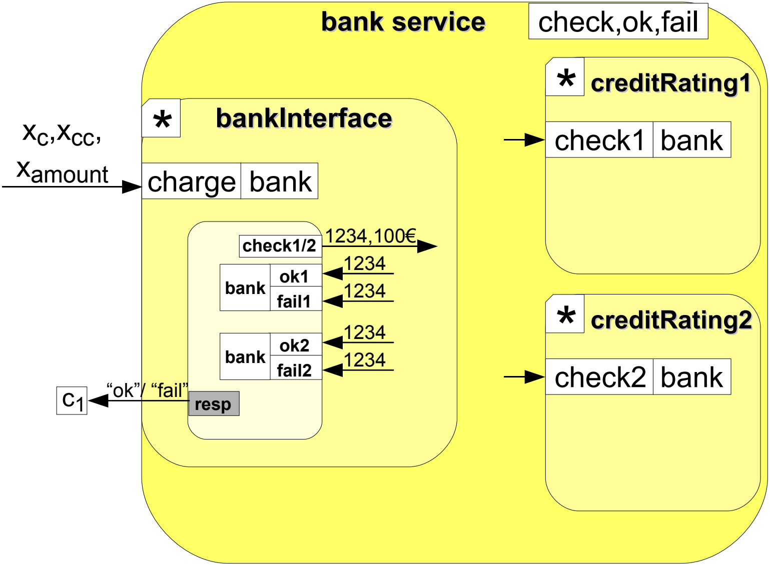
$$\frac{s \xrightarrow{\mathrm{n}\, \sigma \uplus \{x \mapsto v\}\, \ell\, \bar{v}} s'}{[x]\, s \xrightarrow{\mathrm{n}\, \sigma\, \ell\, \bar{v}} s' \cdot \{x \mapsto v\}} \qquad \frac{s_1 \xrightarrow{\mathrm{n}\, \sigma\, \ell\, \bar{v}} s_1' \quad \mathrm{noConf}(s_2, \mathrm{n}, \bar{v}, \ell)}{s_1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

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

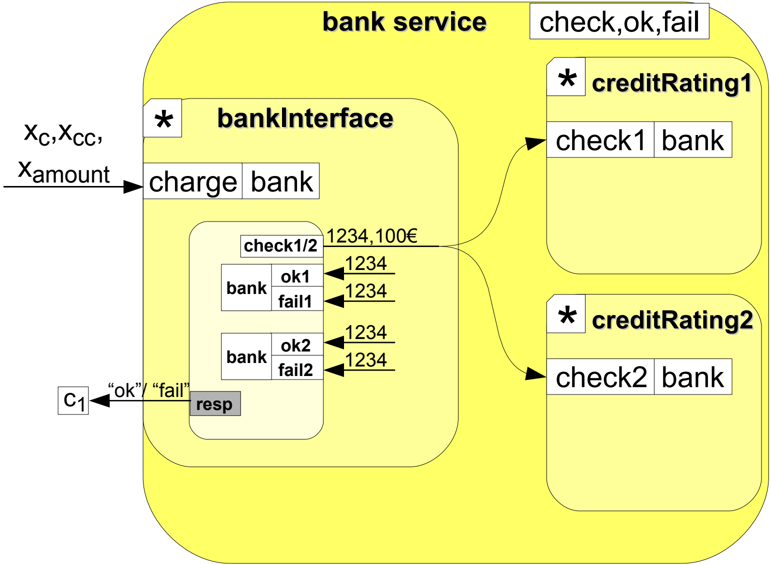
$$\frac{s \xrightarrow{k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'} \qquad \frac{s \xrightarrow{k} s' \quad k \neq e}{[e]\, s \xrightarrow{k} [e]\, s'} \qquad \frac{s_1 \xrightarrow{k} s_1'}{s_1 \mid s_2 \xrightarrow{k} s_1' \mid \mathrm{halt}(s_2)}$$

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

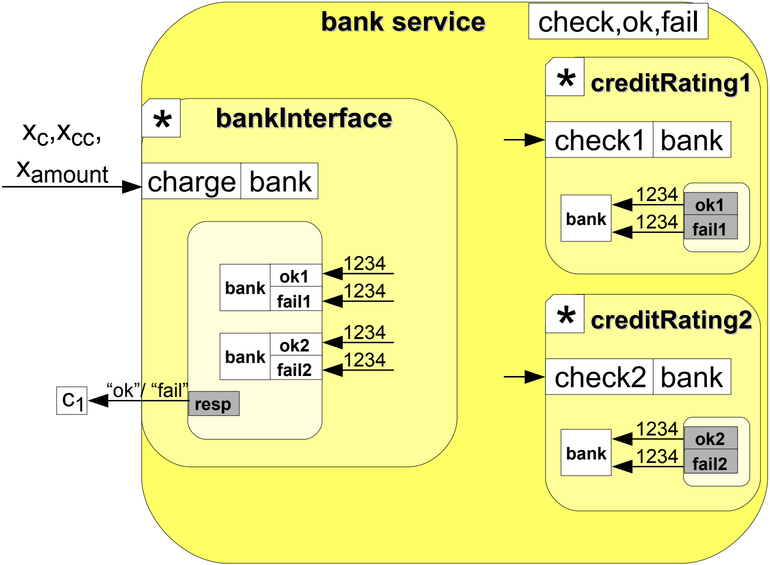
COWS: *multi rating* bank service example



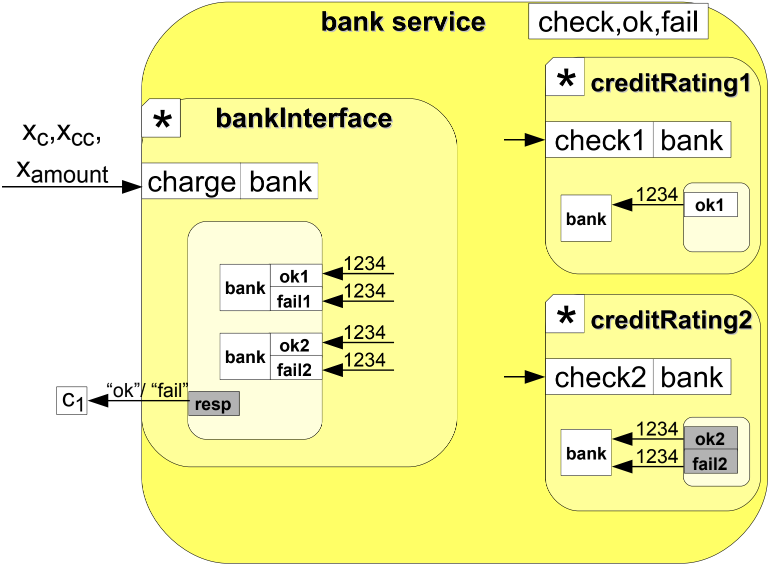
COWS: *multi rating* bank service example



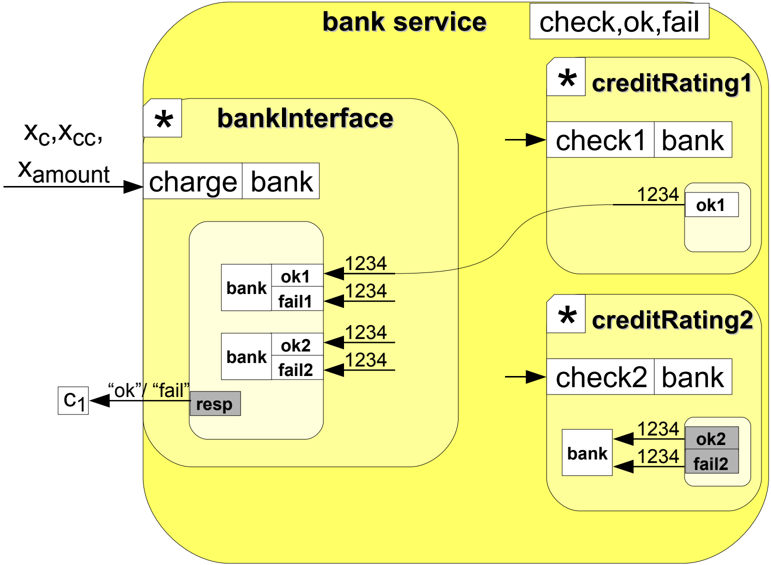
COWS: *multi rating* bank service example



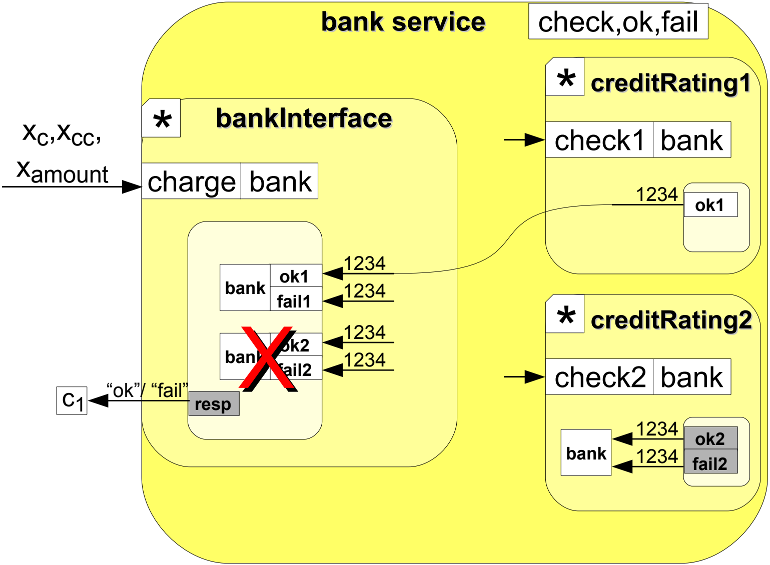
COWS: *multi rating* bank service example



COWS: *multi rating* bank service example



COWS: *multi rating* bank service example



COWS: *multi rating* bank service example

$$[\mathsf{check1}, \mathsf{check2}, \mathsf{ok1}, \mathsf{ok2}, \mathsf{fail1}, \mathsf{fail2}]$$
$$(\, * \,\mathsf{bankInterface} \mid * \,\mathsf{creditRating1} \mid * \,\mathsf{creditRating2}\,)$$

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

COWS: peculiar examples

Protected kill activity

- Execution of a kill activity within a protection block

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

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

- $\mathbf{kill}(k)$ 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 $\mathrm{halt}(s_1) = \mathrm{halt}(s_3) = \mathbf{0}$

- $\mathbf{kill}(k)$ 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 $\mathrm{halt}(s_1) = \mathrm{halt}(s_3) = \mathbf{0}$

- $\mathbf{kill}(k)$ 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

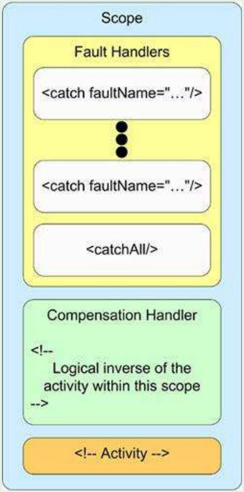
- Encoding other calculi
  - π-calculus, Localized π-calculus (Lπ), ...
  - SCC (Session Centered Calculus)
  - Orc
  - WS-CALCULUS
  - B*lite* (a lightweight version of WS-BPEL)

- 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



COWS: fault and compensation handling

Syntax for compensation

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

- $\textbf{throw}(\phi)$: rises a fault signal ϕ that triggers execution of s if a construct $\textbf{catch}(\phi)\{s\}$ exists within the same scope
- $\textbf{compensate}(i)$: 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

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

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

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

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

Analysis techniques

Analysis techniques for COWS specifications

1. A bisimulation-based observational semantics [ICALP'09]
2. A type system for checking confidentiality properties [FSEN'07]
3. A logical verification methodology [FASE'08]

Analysis techniques: an observational semantics

Behavioural equivalences: key concepts

- An important ingredient of a process calculus is a notion of behavioural equivalences between its terms

- Behavioural equivalences, and the related proof techniques, are a tool providing a means to establishing formal correspondences between terms of a process calculus

- Syntactically different terms may behave the same way, hence they ought to be considered behaviourally equivalent

- Behavioural equivalences can take into account diverse *observable* properties of terms (name mobility, asynchrony, . . .)
  - Several different classes of behavioural equivalences have been introduced, each one being characterised by a specific notion of *observable behaviour*
  - The semantics induced by such equivalences are indeed called *observational semantics*

Behavioural equivalences: key concepts

- An important ingredient of a process calculus is a notion of behavioural equivalences between its terms
- Behavioural equivalences, and the related proof techniques, are a tool providing a means to establishing formal correspondences between terms of a process calculus
- Syntactically different terms may behave the same way, hence they ought to be considered behaviourally equivalent
- Behavioural equivalences can take into account diverse *observable* properties of terms (name mobility, asynchrony, ...)
  - Several different classes of behavioural equivalences have been introduced, each one being characterised by a specific notion of *observable behaviour*
  - The semantics induced by such equivalences are indeed called *observational semantics*

Behavioural equivalences: key concepts

- An important ingredient of a process calculus is a notion of behavioural equivalences between its terms
- Behavioural equivalences, and the related proof techniques, are a tool providing a means to establishing formal correspondences between terms of a process calculus
- Syntactically different terms may behave the same way, hence they ought to be considered behaviourally equivalent
- Behavioural equivalences can take into account diverse *observable* properties of terms (name mobility, asynchrony, . . .)
  - Several different classes of behavioural equivalences have been introduced, each one being characterised by a specific notion of *observable behaviour*
  - The semantics induced by such equivalences are indeed called *observational semantics*

Behavioural equivalences: key concepts

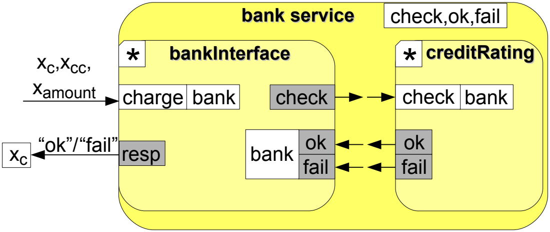
- Powerful and widespread used techniques are based on the notion of *bisimulation*
- Intuitively, a bisimulation is a relation that permits associating two terms if one simulates the behaviour (i.e. the actions that can be performed) of the other and vice-versa
- In doing this, the behaviour of intermediate states that the terms traverse as they evolve have taken into account
  - The action capabilities of the intermediate states does matter: e.g. to observe different deadlock behaviours

An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications

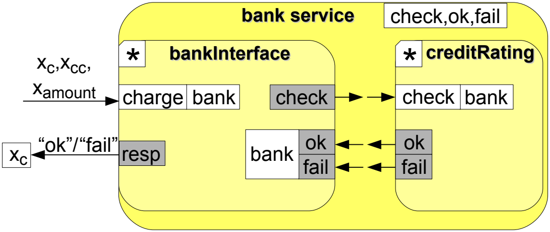
An observational semantics for COWS

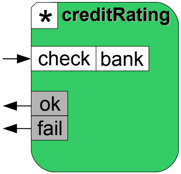
An *observational semantics* permits checking interchangeability of services and conformance against service specifications



An observational semantics for COWS

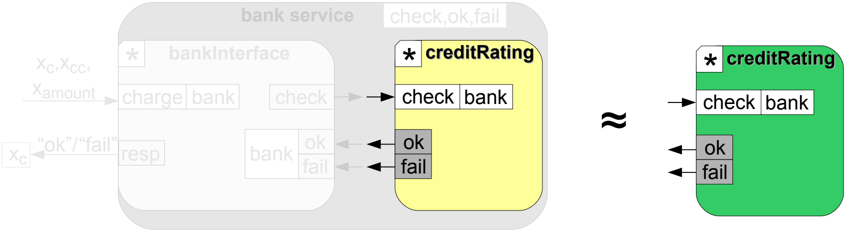
An *observational semantics* permits checking interchangeability of services and conformance against service specifications





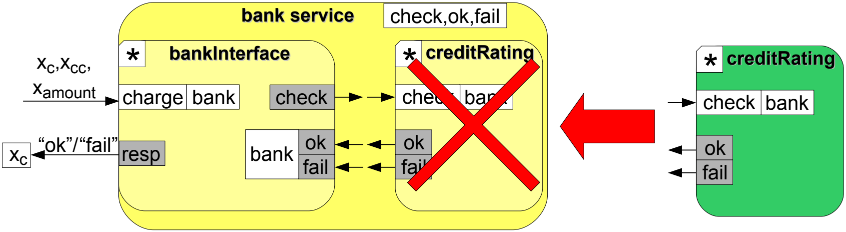
An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications



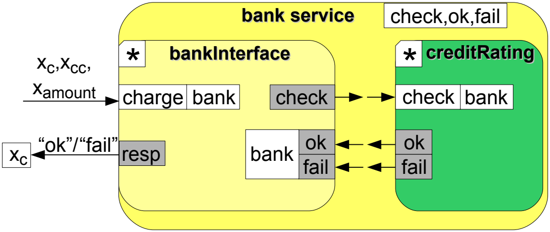
An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications



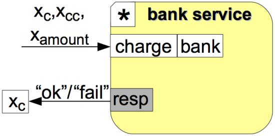
An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications

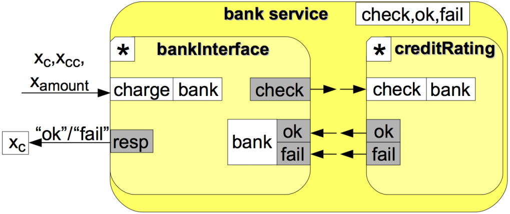


An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications



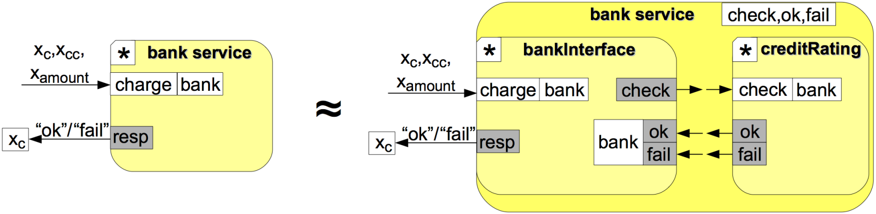
high-level spec.



low-level spec.

An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications



high-level spec. low-level spec.

An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications

- We have defined:
  - natural notions of strong and weak *open barbed bisimilarities*
  - manageable characterisations in terms of *labelled bisimilarities*
- These semantics show that:
  - COWS's priority mechanisms partially recover the capability to observe receive actions
  - primitives for termination impose specific conditions on the bisimilarities

A natural notion of bisimulation

Observable (barb)

Predicate $s \downarrow_{\mathrm{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ s.t. $s \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s'$, i.e. only the output capabilities are considered as observable

E.g. $[\bar{x}]\,(\mathrm{n}?\bar{x} \mid \mathrm{n}!\bar{v}) \downarrow_{\mathrm{n}}$, while $[\bar{x}]\,(\mathrm{n}?\bar{x}) \not\downarrow_{\mathrm{n}}$

Barbed bisimilarity $\simeq$

is the largest symmetric, barb preserving, computation and context closed relation over COWS terms

- Barbed bisimilarity suffers from universal quantification over all possible language contexts
  - this makes the reasoning on terms very hard
- We have provided a purely co-inductive notion of bisimulation
  - only requires considering transitions of the labelled transition system defining the semantics of the terms under analysis

A natural notion of bisimulation

Observable (barb)

Predicate $s \downarrow_{\mathrm{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ s.t. $s \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s'$,
i.e. only the output capabilities are considered as observable

E.g. $[\bar{x}]\,(\mathrm{n}?\bar{x} \mid \mathrm{n}!\bar{v}) \downarrow_{\mathrm{n}}$, while $[\bar{x}]\,(\mathrm{n}?\bar{x}) \not\downarrow_{\mathrm{n}}$

Barbed bisimilarity $\simeq$

is the largest symmetric, barb preserving, computation and context closed relation over COWS terms

- Barbed bisimilarity suffers from universal quantification over all possible language contexts
  - this makes the reasoning on terms very hard
- We have provided a purely co-inductive notion of bisimulation
  - only requires considering transitions of the labelled transition system defining the semantics of the terms under analysis

A natural notion of bisimulation

Observable (barb)

Predicate $s \downarrow_{\mathrm{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ s.t. $s \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s'$, i.e. only the output capabilities are considered as observable

E.g. $[\bar{x}]\,(\mathrm{n}?\bar{x} \mid \mathrm{n}!\bar{v}) \downarrow_{\mathrm{n}}$, while $[\bar{x}]\,(\mathrm{n}?\bar{x}) \not\downarrow_{\mathrm{n}}$

Barbed bisimilarity $\simeq$

is the largest symmetric, barb preserving, computation and context closed relation over COWS terms

- Barbed bisimilarity suffers from universal quantification over all possible language contexts
  - this makes the reasoning on terms very hard
- We have provided a purely co-inductive notion of bisimulation
  - only requires considering transitions of the labelled transition system defining the semantics of the terms under analysis

A natural notion of bisimulation

Observable (barb)

Predicate $s \downarrow_{\mathrm{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ s.t. $s \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s'$, i.e. only the output capabilities are considered as observable

E.g. $[\bar{x}]\,(\mathrm{n}?\bar{x} \mid \mathrm{n}!\bar{v}) \downarrow_{\mathrm{n}}$, while $[\bar{x}]\,(\mathrm{n}?\bar{x}) \not\downarrow_{\mathrm{n}}$

Barbed bisimilarity $\simeq$

is the largest symmetric, barb preserving, computation and context closed relation over COWS terms

- Barbed bisimilarity suffers from universal quantification over all possible language contexts
  - this makes the reasoning on terms very hard
- We have provided a purely co-inductive notion of bisimulation
  - only requires considering transitions of the labelled transition system defining the semantics of the terms under analysis

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

A names-indexed family of symmetric binary relations $\{\mathcal{R}_\mathcal{N}\}_\mathcal{N}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_\mathcal{N} s_2$ then $\mathrm{halt}(s_1)\, \mathcal{R}_\mathcal{N}\, \mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if $\alpha = \mathrm{n} \triangleright [\bar{x}]\, \bar{w}$ then one of the following holds:
   (a) $\forall\, \bar{v}$ s.t. $\mathcal{M}(\bar{x}, \bar{v}) = \sigma$ and $noc(s_2, \mathrm{n}, \bar{w}\cdot\sigma, |\,\bar{x}\,|)$:
   $\exists\, s_2' \ :\ s_2 \xrightarrow{\mathrm{n}\triangleright[\bar{x}]\,\bar{w}} s_2'$ and $s_1'\cdot\sigma\, \mathcal{R}_\mathcal{N}\, s_2'\cdot\sigma$
   (b) $|\,\bar{x}\,| = |\,\bar{w}\,|$ and $\forall\, \bar{v}$ s.t. $\mathcal{M}(\bar{x}, \bar{v}) = \sigma$ and $noc(s_2, \mathrm{n}, \bar{w}\cdot\sigma, |\,\bar{x}\,|)$:
   $\exists\, s_2' \ :\ s_2 \xrightarrow{\emptyset} s_2'$ and $s_1'\cdot\sigma\, \mathcal{R}_\mathcal{N}\, (s_2' \mid \mathrm{n}!\bar{v})$ or $s_1'\cdot\sigma\, \mathcal{R}_\mathcal{N}\, (s_2' \mid \{\!|\mathrm{n}!\bar{v}|\!\})$
2. if $\alpha = \mathrm{n}\,\emptyset\,\ell\,\bar{v}$ where $\ell = |\,\bar{v}\,|$ then one of the following holds:
   (a) $\exists\, s_2' \ :\ s_2 \xrightarrow{\mathrm{n}\,\emptyset\,\ell\,\bar{v}} s_2'$ and $s_1'\, \mathcal{R}_\mathcal{N}\, s_2'$ (b) $\exists\, s_2' \ :\ s_2 \xrightarrow{\emptyset} s_2'$ and $s_1'\, \mathcal{R}_\mathcal{N}\, s_2'$
3. if $\alpha = \mathrm{n} \triangleleft [\bar{n}]\, \bar{v}$ where $\mathrm{n} \notin \mathcal{N}$ then $\exists\, s_2' \ :\ s_2 \xrightarrow{\mathrm{n}\triangleleft[\bar{n}]\,\bar{v}} s_2'$ and $s_1'\, \mathcal{R}_{\mathcal{N}\cup\bar{n}}\, s_2'$
4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathrm{n}\,\emptyset\,\ell\,\bar{v}$, where $\ell \neq |\,\bar{v}\,|$, then $\exists\, s_2' \ :\ s_2 \xrightarrow{\alpha} s_2'$ and $s_1'\, \mathcal{R}_\mathcal{N}\, s_2'$

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^\mathcal{N} s_2$, if $s_1 \mathcal{R}_\mathcal{N} s_2$ for some $\mathcal{R}_\mathcal{N}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^\mathcal{N}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

A names-indexed family of symmetric binary relations $\{\mathcal{R}_{\mathcal{N}}\}_{\mathcal{N}}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ then $\mathrm{halt}(s_1)\, \mathcal{R}_{\mathcal{N}}\, \mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if s_1 performs a *receive* then one of the following holds:
   - (a) s_2 performs the same receive and the continuations stand in the same relation for any matching tuple of values that can be effectively received
   - (b) if the argument of the receive contains only variables or is the empty tuple, s_2 performs an internal action leading to a term that, composed with the consumed invoke, stands in the same relation
2. if $\alpha = \mathrm{n}\,\emptyset\,\ell\,\bar{v}$ where $\ell = \mid \bar{v} \mid$ then one of the following holds:
   (a) $\exists\, s_2' \,:\, s_2 \xrightarrow{\mathrm{n}\,\emptyset\,\ell\,\bar{v}} s_2'$ and $s_1' \mathcal{R}_{\mathcal{N}}\, s_2'$ (b) $\exists\, s_2' \,:\, s_2 \xrightarrow{\emptyset} s_2'$ and $s_1' \mathcal{R}_{\mathcal{N}}\, s_2'$
3. if $\alpha = \mathrm{n} \lhd [\bar{n}]\, \bar{v}$ where $\mathrm{n} \notin \mathcal{N}$ then $\exists\, s_2' \,:\, s_2 \xrightarrow{\mathrm{n}\lhd[\bar{n}]\,\bar{v}} s_2'$ and $s_1' \mathcal{R}_{\mathcal{N}\cup\bar{n}}\, s_2'$
4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathrm{n}\,\emptyset\,\ell\,\bar{v}$, where $\ell \neq \mid \bar{v} \mid$, then $\exists\, s_2' \,:\, s_2 \xrightarrow{\alpha} s_2'$ and $s_1' \mathcal{R}_{\mathcal{N}}\, s_2'$

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ for some $\mathcal{R}_{\mathcal{N}}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

A names-indexed family of symmetric binary relations $\{\mathcal{R}_{\mathcal{N}}\}_{\mathcal{N}}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ then $\mathrm{halt}(s_1)\, \mathcal{R}_{\mathcal{N}}\, \mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if s_1 performs a *receive* then one of the following holds:
   (a) s_2 performs the same receive and the continuations stand in the same relation for any matching tuple of values that can be effectively received
   (b) if the argument of the receive contains only variables or is the empty tuple, s_2 performs an internal action leading to a term that, composed with the consumed invoke, stands in the same relation
2. if s_1 performs a *communication* involving an unobservable receive then: s_2 performs (a) the same action or (b) an internal action and . . .
3. if $\alpha = \mathrm{n} \lhd [\bar{n}]\, \bar{v}$ where $\mathrm{n} \notin \mathcal{N}$ then $\exists\, s_2' \;:\; s_2 \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s_2'$ and $s_1' \, \mathcal{R}_{\mathcal{N} \cup \bar{n}}\, s_2'$
4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathrm{n}\, \emptyset\, \ell\, \bar{v}$, where $\ell \neq |\, \bar{v}\, |$, then $\exists\, s_2' \;:\; s_2 \xrightarrow{\alpha} s_2'$ and $s_1' \, \mathcal{R}_{\mathcal{N}}\, s_2'$

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ for some $\mathcal{R}_{\mathcal{N}}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

A names-indexed family of symmetric binary relations $\{\mathcal{R}_{\mathcal{N}}\}_{\mathcal{N}}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ then $\mathrm{halt}(s_1)\,\mathcal{R}_{\mathcal{N}}\,\mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if s_1 performs a *receive* then one of the following holds:
   (a) s_2 performs the same receive and the continuations stand in the same relation for any matching tuple of values that can be effectively received
   (b) if the argument of the receive contains only variables or is the empty tuple, s_2 performs an internal action leading to a term that, composed with the consumed invoke, stands in the same relation
2. if s_1 performs a *communication* involving an unobservable receive then: s_2 performs (a) the same action or (b) an internal action and ...
3. if s_1 performs an *invoke* then s_2 performs the same invoke and ...
4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathrm{n}\,\emptyset\,\ell\,\bar{v}$, where $\ell \neq |\,\bar{v}\,|$, then $\exists\, s_2' \;:\; s_2 \xrightarrow{\alpha} s_2'$ and $s_1' \,\mathcal{R}_{\mathcal{N}}\, s_2'$

Two closed terms s_1 and s_2 are *$\mathcal{N}$-bisimilar*, written $s_1 \sim^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ for some $\mathcal{R}_{\mathcal{N}}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

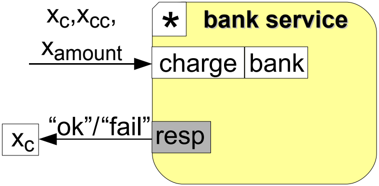
A names-indexed family of symmetric binary relations $\{\mathcal{R}_\mathcal{N}\}_\mathcal{N}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_\mathcal{N} s_2$ then $\mathrm{halt}(s_1)\, \mathcal{R}_\mathcal{N}\, \mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if s_1 performs a *receive* then one of the following holds:
   (a) s_2 performs the same receive and the continuations stand in the same relation for any matching tuple of values that can be effectively received
   (b) if the argument of the receive contains only variables or is the empty tuple, s_2 performs an internal action leading to a term that, composed with the consumed invoke, stands in the same relation
2. if s_1 performs a *communication* involving an unobservable receive then: s_2 performs (a) the same action or (b) an internal action and ...
3. if s_1 performs an *invoke* then s_2 performs the same invoke and ...
4. if s_1 performs either an *internal action*, a *kill* or a *communication* involving an observable receive then s_2 performs the same action and ...

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^\mathcal{N} s_2$, if $s_1 \mathcal{R}_\mathcal{N} s_2$ for some $\mathcal{R}_\mathcal{N}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^\mathcal{N}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

Observational semantics at work: bank service

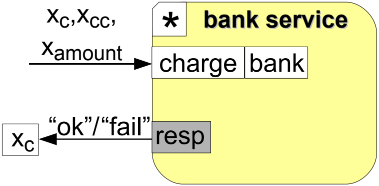
We can compare the high-level specification



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

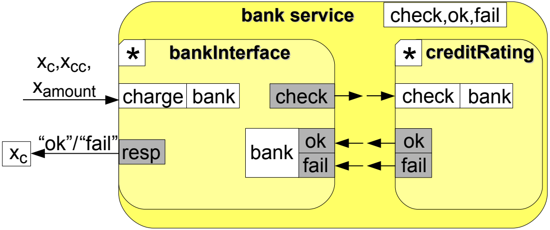
Observational semantics at work: bank service

We can compare the high-level specification



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

with the low-level specification

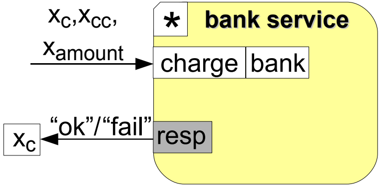


$$
\begin{array}{l}
[\mathrm{check}, \mathrm{ok}, \mathrm{fail}] \\
(\, * \, \mathrm{bankInterface} \mid * \, \mathrm{creditRating} \,)
\end{array}
$$

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

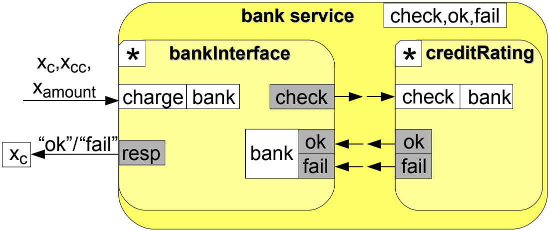
Observational semantics at work: bank service

We can compare the high-level specification



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

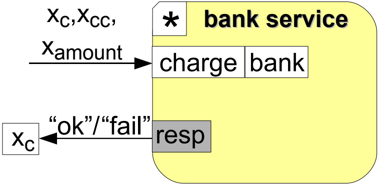
with the low-level specification



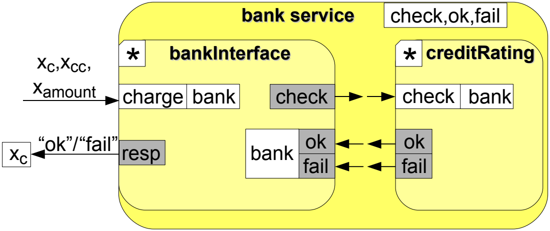
$$[\mathrm{check}, \mathrm{ok}, \mathrm{fail}]$$
$$(* \mathrm{bankInterface} \mid * \mathrm{creditRating})$$

$$\mathrm{creditRating} \triangleq$$
$$[\mathrm{x_{cc}}, \mathrm{x_a}]$$
$$\mathrm{bank} \bullet \mathrm{check}?\langle \mathrm{x_{cc}}, \mathrm{x_a} \rangle .$$
$$[\mathrm{p}, \mathrm{o}] \, (\mathrm{p} \bullet \mathrm{o}!\langle chk(x_{cc}, x_{amount}) \rangle$$
$$\mid \mathrm{p} \bullet \mathrm{o}?\langle \text{“ok”} \rangle . \mathrm{bank} \bullet \mathrm{ok}!\langle \mathrm{x_{cc}} \rangle$$
$$+ \mathrm{p} \bullet \mathrm{o}?\langle \text{“fail”} \rangle . \mathrm{bank} \bullet \mathrm{fail}!\langle \mathrm{x_{cc}} \rangle \,)$$

Observational semantics at work: bank service



$\approx$



Examples: observation of receive actions

Asynchronous π-calculus: the input absorption law

$$\tau + a(b).\,\bar{a}b \;\sim\; \tau$$

COWS without priority: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\sim\; \emptyset$$

where $\emptyset \triangleq [p', o']\,(p' \bullet o'!\langle\rangle \mid p' \bullet o'?\langle\rangle)$

COWS: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\not\sim\; \emptyset$$

since $\mathbb{C} \triangleq [y, z]\, p \bullet o?\langle y, z\rangle.\, p'' \bullet o''!\langle\rangle \mid p \bullet o!\langle v', v\rangle \mid [\![\cdot]\!]$ can distinguish them

However

$$[x, y]\,(\,\emptyset + p \bullet o?\langle x, y\rangle.\, p \bullet o!\langle x, y\rangle\,) \;\sim\; \emptyset$$

Examples: observation of receive actions

Asynchronous π-calculus: the input absorption law

$$\tau + a(b).\,\bar{a}b \;\sim\; \tau$$

COWS without priority: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\sim\; \emptyset$$

where $\emptyset \triangleq [p', o']\,(p' \bullet o'!\langle\rangle \mid p' \bullet o'?\langle\rangle)$

COWS: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\not\sim\; \emptyset$$

since $\mathbb{C} \triangleq [y, z]\, p \bullet o?\langle y, z\rangle.\, p'' \bullet o''!\langle\rangle \mid p \bullet o!\langle v', v\rangle \mid [\![\cdot]\!]$ can distinguish them

However

$$[x, y]\,(\,\emptyset + p \bullet o?\langle x, y\rangle.\, p \bullet o!\langle x, y\rangle\,) \;\sim\; \emptyset$$

Examples: observation of receive actions

Asynchronous π-calculus: the input absorption law

$$\tau + a(b).\, \bar{a}b \;\sim\; \tau$$

COWS without priority: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\sim\; \emptyset$$

where $\emptyset \triangleq [p', o']\,(p' \bullet o'!\langle\rangle \mid p' \bullet o'?\langle\rangle)$

COWS: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\not\sim\; \emptyset$$

since $\mathbb{C} \triangleq [y, z]\, p \bullet o?\langle y, z\rangle.\, p'' \bullet o''!\langle\rangle \mid p \bullet o!\langle v', v\rangle \mid [\![\cdot]\!]$ can distinguish them

However

$$[x, y]\,(\,\emptyset + p \bullet o?\langle x, y\rangle.\, p \bullet o!\langle x, y\rangle\,) \;\sim\; \emptyset$$

Examples: observation of receive actions

Asynchronous π-calculus: the input absorption law

$$\tau + a(b).\, \bar{a}b \sim \tau$$

COWS without priority: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \sim \emptyset$$

where $\emptyset \triangleq [p', o']\,(p' \bullet o'!\langle\rangle \mid p' \bullet o'?\langle\rangle)$

COWS: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \not\sim \emptyset$$

since $\mathbb{C} \triangleq [y, z]\, p \bullet o?\langle y, z\rangle.\, p'' \bullet o''!\langle\rangle \mid p \bullet o!\langle v', v\rangle \mid [\![\cdot]\!]$ can distinguish them

However

$$[x, y]\,(\,\emptyset + p \bullet o?\langle x, y\rangle.\, p \bullet o!\langle x, y\rangle\,) \sim \emptyset$$

Analysis techniques: a type system

A type system for confidentiality properties

- Type systems could be a scalable way to provide evidence that a large number of SOC applications enjoy some given properties

Confidentiality properties

Critical data (e.g. credit card information) are shared only by authorized partners

- Our type system permits
  - expressing and forcing policies regulating the exchange of data among interacting services
  - ensuring that, in that respect, services do not manifest unexpected behaviours

Syntax of typed COWS

$s ::=$		(services)
	$\mathbf{kill}(k)$	(kill)
$\mid$	$u \bullet u'!\langle \{\epsilon_1\}_{r_1}, \ldots, \{\epsilon_n\}_{r_n} \rangle$	(invoke)
$\mid$	$\sum_{i=0}^{r} p_i \bullet o_i?\bar{w}_i.s_i$	(choice)
$\mid$	$s \mid s$	(parallel)
$\mid$	$\{\!\|s\|\!\}$	(protection)
$\mid$	$[e]\, s$	(delimitation)
$\mid$	$* \, s$	(replication)

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

Programmers can settle the partners usable to exchange any given datum, thus avoiding the datum be accessed by unwanted services

- Data are annotated with *regions*: $u \bullet u'!\langle \{\epsilon_1\}_{r_1}, \ldots, \{\epsilon_n\}_{r_n} \rangle$
- Regions $r_1 \ldots r_n$ specify the policies regulating the exchange of the data resulting from evaluation of $\epsilon_1 \ldots \epsilon_n$
- A region r can be either a finite subset of partners and variables or the distinct element $\top$ (denoting the universe of partners)

Syntax of typed COWS

$s ::=$		(services)
	$\mathbf{kill}(k)$	(kill)
$\mid$	$u \bullet u'!\langle \{\epsilon_1\}_{r_1}, \ldots, \{\epsilon_n\}_{r_n} \rangle$	(invoke)
$\mid$	$\sum_{i=0}^{r} p_i \bullet o_i?\bar{w}_i.s_i$	(choice)
$\mid$	$s \mid s$	(parallel)
$\mid$	$\{\!\mid s \mid\!\}$	(protection)
$\mid$	$[e]\, s$	(delimitation)
$\mid$	$* \, s$	(replication)

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

Programmers can settle the partners usable to exchange any given datum, thus avoiding the datum be accessed by unwanted services

- Data are annotated with *regions*: $u \bullet u'!\langle \{\epsilon_1\}_{r_1}, \ldots, \{\epsilon_n\}_{r_n} \rangle$
- Regions $r_1 \ldots r_n$ specify the policies regulating the exchange of the data resulting from evaluation of $\epsilon_1 \ldots \epsilon_n$
- A region r can be either a finite subset of partners and variables or the distinct element $\top$ (denoting the universe of partners)

Static and dynamic semantics

Static semantics

A static type system infers region annotations for variable declarations and returns well-typed terms

Dynamic semantics

The operational semantics exploits region annotations to authorize or block the exchange of data

Static semantic

- The *static* type inference system has two main tasks
  - performs some coherence checks
    e.g. the partner used by an invoke must belong to the regions of all data occurring in the argument of the activity
  - derives the minimal region annotations for variable declarations that ensure consistency of services initial configuration
    - $[\{x\}^r]\, s$ means that the datum that dynamically will replace x will be used at most by the partners in r

- Typing judgements are written $\Gamma \vdash s \succ \Gamma' \vdash s'$, where the type environment Γ is a finite function from variables to regions

- s is *well-typed* if $\emptyset \vdash s' \succ \emptyset \vdash s$, for some s'
  i.e. s is the (typed) service obtained by decorating s' with the regions describing the use of each variable of s' in its scope

Static semantic

- The *static* type inference system has two main tasks
  - performs some coherence checks
    e.g. the partner used by an invoke must belong to the regions of all data occurring in the argument of the activity
  - derives the minimal region annotations for variable declarations that ensure consistency of services initial configuration
    - $[\{x\}^r]\, s$ means that the datum that dynamically will replace x will be used at most by the partners in r

- Typing judgements are written $\Gamma \vdash s \succ \Gamma' \vdash s'$, where the type environment Γ is a finite function from variables to regions

- s is *well-typed* if $\emptyset \vdash s' \succ \emptyset \vdash s$, for some s'
  i.e. s is the (typed) service obtained by decorating s' with the regions describing the use of each variable of s' in its scope

Static semantics : significant typing rules

- Rule for (monadic) invoke activity:

$$\frac{u \in r}{\Gamma \;\vdash\; u \bullet u'!\{e(\bar{y})\}_r \;\succ\; (\Gamma + \{x : r\}_{x \in \bar{y}}) \;\vdash\; u \bullet u'!\{e(\bar{y})\}_r}$$

  - it checks if the invoked partner u belongs to the region of the datum
  - if it succeeds, the type environment Γ is extended by associating a proper region to each variable used in the argument expression e

- Rule for variable delimitation:

$$\frac{\Gamma \uplus \{x : \emptyset\} \;\vdash\; s \;\succ\; \Gamma' \uplus \{x : r\} \;\vdash\; s' \qquad x \notin \mathrm{reg}(\Gamma')}{\Gamma \;\vdash\; [x]\, s \;\succ\; \Gamma' \;\vdash\; [\{x\}^{r-\{x\}}]\, s'}$$

  - it annotates the delimitation with the region associated to it by the type environment
  - premiss $x \notin \mathrm{reg}(\Gamma')$ and annotation $r - \{x\}$ prevent initially closed services to become open at the end of the inference

Static semantics : significant typing rules

- Rule for (monadic) invoke activity:

$$\frac{u \in r}{\Gamma \;\vdash\; u \bullet u'!\{e(\bar{y})\}_r \;\succ\; (\Gamma + \{x : r\}_{x \in \bar{y}}) \;\vdash\; u \bullet u'!\{e(\bar{y})\}_r}$$

  - it checks if the invoked partner u belongs to the region of the datum
  - if it succeeds, the type environment Γ is extended by associating a proper region to each variable used in the argument expression e

- Rule for variable delimitation:

$$\frac{\Gamma \uplus \{x : \emptyset\} \;\vdash\; s \;\succ\; \Gamma' \uplus \{x : r\} \;\vdash\; s' \qquad x \notin \mathrm{reg}(\Gamma')}{\Gamma \;\vdash\; [x]\, s \;\succ\; \Gamma' \;\vdash\; [\{x\}^{r-\{x\}}]\, s'}$$

  - it annotates the delimitation with the region associated to it by the type environment
  - premiss $x \notin \mathrm{reg}(\Gamma')$ and annotation $r - \{x\}$ prevent initially closed services to become open at the end of the inference

Dynamic semantics

- The language operational semantics only performs efficiently implementable checks to authorize or block communication
  - types are just sets of (partner) names
  - the region annotation (policy) of output data must contain the region annotation of the corresponding input variables

- The most significant modified rule:

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

Dynamic semantics

- The language operational semantics only performs efficiently implementable checks to authorize or block communication
  - types are just sets of (partner) names
  - the region annotation (policy) of output data must contain the region annotation of the corresponding input variables

- The most significant modified rule:

$$\frac{s \xrightarrow{\mathtt{n}\,\sigma \uplus \{x \mapsto \{v\}_r\}\,\ell\,\bar{v}} s' \qquad r' \cdot \sigma \subseteq r}{[\{x\}^{r'}]\, s \xrightarrow{\mathtt{n}\,\sigma\,\ell\,\bar{v}} s' \cdot \{x \mapsto \{v\}_r\}}$$

Results

Major results

Subject reduction & *type safety* results imply that services always comply with the constraints (expressed by the type) of each datum

- *Subject reduction* states that well-typedness is preserved along computations
- *Type safety* states that well-typed services do respect region annotations

Results

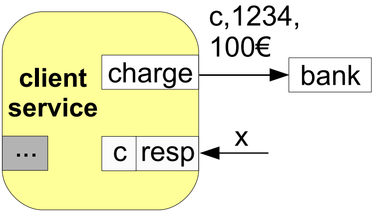
Major results

Subject reduction & *type safety* results imply that services always comply with the constraints (expressed by the type) of each datum

Soundness

A service *s* is *sound* if, for any datum *v* occurring in *s* associated to region *r* and for all possible evolutions of *s*, it holds that *v* can only be exchanged using partners in *r*

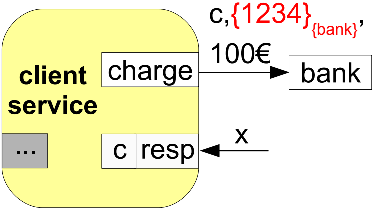
The bank service with security policies



client $\triangleq$
$\quad$ bank $\bullet$ charge!$\langle$c, 1234, 100€$\rangle$
$\quad$ | [x] (c $\bullet$ resp?$\langle$x$\rangle$.s | s′)

Client policy : *only* bank is authorized to access credit card data

The bank service with security policies

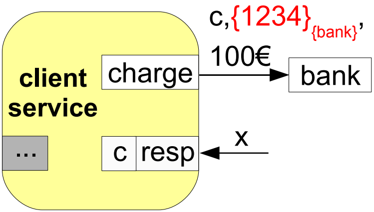


Tclient $\triangleq$

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

Client policy : *only* bank is authorized to access credit card data

The bank service with security policies



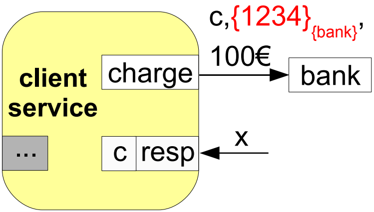
Tclient $\triangleq$

bank • charge!⟨c, {1234}$_{\{bank\}}$, 100€⟩
| [x] (c • resp?⟨x⟩.s | s′)

Client policy : *only* bank is authorized to access credit card data

The type system infers the region annotations for the bank service, e.g. . . .

The bank service with security policies

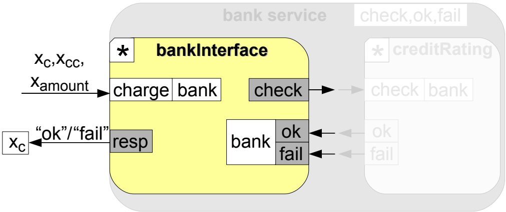


Tclient $\triangleq$

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

Client policy : *only* bank is authorized to access credit card data

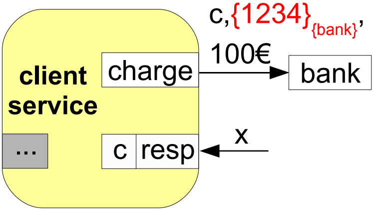
The type system infers the region annotations for the bank service, e.g. . . .



bankInterface $\triangleq$

$$[x_c, x_{cc}, x_{amount}]\; \mathsf{bank} \bullet \mathsf{charge}?\langle x_c, x_{cc}, x_{amount} \rangle .\; (\, \mathsf{bank} \bullet \mathsf{check}!\langle x_{cc}, x_{amount} \rangle \;|\; \mathsf{bank} \bullet \mathsf{ok}?\langle x_{cc} \rangle .\, x_c \bullet \mathsf{resp}!\langle \text{“ok”} \rangle + \mathsf{bank} \bullet \mathsf{fail}?\langle x_{cc} \rangle .\, x_c \bullet \mathsf{resp}!\langle \text{“fail”} \rangle \,)$$

The bank service with security policies

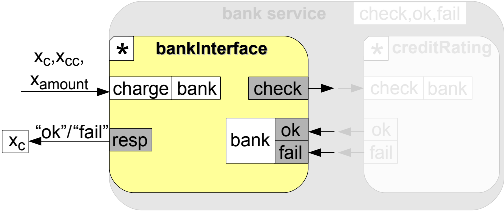


Tclient $\triangleq$

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

Client policy : *only* bank is authorized to access credit card data

The type system infers the region annotations for the bank service, e.g. . . .



TbankInterface $\triangleq$

$$[\{\mathsf{x_c}\}^{\{\mathsf{bank}\}}, \{\mathsf{x_{cc}}\}^{\{\mathsf{bank}\}}, \{\mathsf{x_{amount}}\}^{\{\mathsf{bank}\}}]$$
$$\mathsf{bank} \bullet \mathsf{charge}?\langle \mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}} \rangle.$$
$$(\, \mathsf{bank} \bullet \mathsf{check}!\langle \mathsf{x_{cc}}, \mathsf{x_{amount}} \rangle$$
$$\mid \mathsf{bank} \bullet \mathsf{ok}?\langle \mathsf{x_{cc}} \rangle.\, \mathsf{x_c} \bullet \mathsf{resp}!\langle \text{“ok”} \rangle$$
$$+ \mathsf{bank} \bullet \mathsf{fail}?\langle \mathsf{x_{cc}} \rangle.\, \mathsf{x_c} \bullet \mathsf{resp}!\langle \text{“fail”} \rangle \,)$$

The bank service with security policies

$$\begin{array}{lll} \text{Tclient} & \triangleq & \text{bank} \bullet \text{charge!}\langle \text{c}, \{1234\}_{\{\text{bank}\}}, 100€\rangle \\ & & \mid [\text{x}]\,(\,\text{c} \bullet \text{resp?}\langle \text{x}\rangle.\text{s} \mid \text{s}'\,) \end{array}$$

Client policy: *only* bank is authorized to access credit card data

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

By using the statically inferred annotations, . . .

The bank service with security policies

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

Client policy: *only* bank is authorized to access credit card data

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

By using the statically inferred annotations, the operational semantics guarantees that the content of $\mathsf{x_{cc}}$ cannot become available to other services

$$\mathsf{Tclient} \mid [\mathsf{check}, \mathsf{ok}, \mathsf{fail}]\,(\,*\,\mathsf{TbankInterface} \mid *\,\mathsf{creditRating}\,) \longrightarrow \ \ldots$$

Indeed, $\quad \mathsf{region}(\{\mathsf{x_{cc}}\}^{\{\mathsf{bank}\}}) \subseteq \mathsf{region}(\{1234\}_{\{\mathsf{bank}\}})$

A malicious bank service

$$\begin{array}{lcl} \mathsf{Tclient} & \triangleq & \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€\rangle \\ & & \mid [\mathsf{x}]\,(\,\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}\rangle.\mathsf{s} \mid \mathsf{s}'\,) \end{array}$$

Client policy: *only* bank is authorized to access credit card data

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

A malicious bank service

$$\begin{array}{lcl} \text{Tclient} & \triangleq & \text{bank} \bullet \text{charge!}\langle \text{c}, \{1234\}_{\{\text{bank}\}}, 100€\rangle \\ & & \mid [\text{x}]\,(\,\text{c} \bullet \text{resp?}\langle \text{x}\rangle.\text{s} \mid \text{s}'\,) \end{array}$$

Client policy: *only* bank is authorized to access credit card data

$$\begin{array}{lcl} \text{TspyBankInterface} & \triangleq & [\{\text{x}_{\text{c}}\}^{\{\text{bank}\}}, \{\text{x}_{\text{cc}}\}^{\{\text{bank, spy}\}}, \{\text{x}_{\text{amount}}\}^{\{\text{bank, spy}\}}] \\ & & \text{bank} \bullet \text{charge?}\langle \text{x}_{\text{c}}, \text{x}_{\text{cc}}, \text{x}_{\text{amount}}\rangle. \\ & & (\,\text{spy} \bullet \text{check!}\langle \text{x}_{\text{cc}}, \text{x}_{\text{amount}}\rangle \\ & & \;\mid \text{bank} \bullet \text{ok?}\langle \text{x}_{\text{cc}}\rangle.\,\text{x}_{\text{c}} \bullet \text{resp!}\langle\,\text{“ok”}\,\rangle \\ & & \quad + \text{bank} \bullet \text{fail?}\langle \text{x}_{\text{cc}}\rangle.\,\text{x}_{\text{c}} \bullet \text{resp!}\langle\,\text{“fail”}\,\rangle\,) \end{array}$$

From the statically inferred annotations, . . .

A malicious bank service

$$\begin{array}{rcl} \mathsf{Tclient} & \triangleq & \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€\rangle \\ & & \mid [x]\,(\,\mathsf{c} \bullet \mathsf{resp}?\langle x\rangle.s \mid s'\,) \end{array}$$

Client policy: *only* bank is authorized to access credit card data

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

From the statically inferred annotations, we can see that the contents of x_{cc} and x_{amount} can become available to spy!

A malicious bank service

$$\begin{array}{rcl} \mathsf{Tclient} & \triangleq & \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€\rangle \\ & & \mid [\mathsf{x}]\,(\,\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}\rangle.\mathsf{s} \mid \mathsf{s}'\,) \end{array}$$

Client policy: *only* bank is authorized to access credit card data

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

From the statically inferred annotations, we can see that the contents of $\mathsf{x_{cc}}$ and $\mathsf{x_{amount}}$ can become available to spy!

The operational semantics does block the transition

$$\mathsf{Tclient} \mid [\mathsf{check}, \mathsf{ok}, \mathsf{fail}]\,(\,*\,\mathsf{TspyBankInterface} \mid *\,\mathsf{creditRating}\,) \not\longrightarrow$$

Indeed, $\quad \mathsf{region}(\{\mathsf{x_{cc}}\}^{\{\mathsf{bank, spy}\}}) \not\subseteq \mathsf{region}(\{1234\}_{\{\mathsf{bank}\}})$

The bank service: a bank policy

$$\begin{array}{rcl} \mathsf{TclientKey} & \triangleq & \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€ \rangle \\ & & \mid [\mathsf{x}, \mathsf{y_{key}}]\,(\,\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}, \mathsf{y_{key}} \rangle.\mathsf{s} \mid \mathsf{s}'\,) \end{array}$$

The client can also receive a personal secret key to be used for successive operations

The bank service: a bank policy

$$\begin{aligned}\mathsf{TclientKey} \;\triangleq\;& \mathsf{bank} \bullet \mathsf{charge}!\langle c, \{1234\}_{\{bank\}}, 100€\rangle \\ & \mid [x, y_{key}]\,(\, c \bullet \mathsf{resp}?\langle x, y_{key}\rangle.s \mid s'\,)\end{aligned}$$

The client can also receive a personal secret key to be used for successive operations

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

Policy: the bank service wants to guarantees that the key sent to the client is not disclosed to third parties

The bank service: a bank policy

$$\begin{array}{lcl} \mathrm{TclientKey} & \triangleq & \mathrm{bank} \bullet \mathrm{charge!}\langle \mathrm{c}, \{1234\}_{\{\mathrm{bank}\}}, 100€\rangle \\ & & \mid [\mathrm{x}, \mathrm{y_{key}}]\,(\,\mathrm{c} \bullet \mathrm{resp?}\langle \mathrm{x}, \mathrm{y_{key}}\rangle.\mathrm{s} \mid \mathrm{s}'\,) \end{array}$$

The client can also receive a personal secret key to be used for successive operations

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

Policy: the bank service wants to guarantees that the key sent to the client is not disclosed to third parties

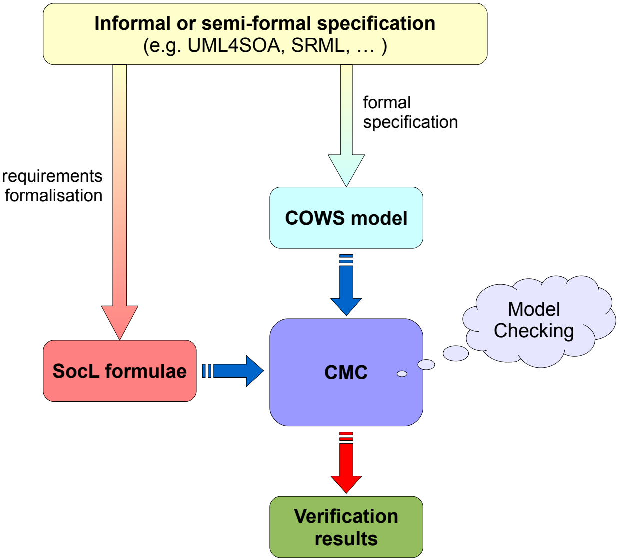
The policy is not fixed at design time, but *depends* on the value of $\mathrm{x_c}$

Analysis techniques: a logical framework

Logics and Model checking

- Process calculi provide behavioral specifications of services
- Logics have been long since proved able to reason about such complex systems as SOC applications
  - provide abstract specifications of these complex systems
  - can be used for describing system properties rather than system behaviors
- Logics and model checkers can be used as tools for verifying that services enjoy desirable properties and do not manifest unexpected behaviors

A logical verification methodology



Requirements formalisation

To formally express service properties we exploit

SocL

an action- and state-based, branching time, temporal logic expressly designed to formalise in a convenient way distinctive aspects of services

action- and state-based logic

$\Downarrow$

Doubly Labelled Transition Systems (L^2TS) as interpretation domain

$\Downarrow$

Abstract notion of services

- services are thought of as sw entities which may have an internal state and can interact with each other
- services are characterised by actions and atomic propositions of the form $type/name(interaction, corrTuple)$

Requirements formalisation

To formally express service properties we exploit

SocL

an action- and state-based, branching time, temporal logic expressly designed to formalise in a convenient way distinctive aspects of services

action- and state-based logic

$\Downarrow$

Doubly Labelled Transition Systems (L^2TS) as interpretation domain

$\Downarrow$

Abstract notion of services

- services are thought of as sw entities which may have an internal state and can interact with each other
- services are characterised by actions and atomic propositions of the form $type/name(interaction, corrTuple)$

SocL actions

Actions ($a \in Act$)

have the form $t(i, c)$

- t: type of the action (e.g. *request*, *response*, *fail*, ...)
- i: name of the interaction which the action is part of (e.g. *charge*)
- c: tuple of correlation values and variables identifying the interaction; $\underline{var}$ denotes a binding occurrence of the correlation variable *var*

Examples

- $request(charge, 1234, 1)$: action starting an (instance of the) interaction *charge* which will be identified through the correlation tuple $\langle 1234, 1 \rangle$

  a corresponding response action can be $response(charge, 1234, 1)$

- $request(charge, 1234, \underline{id})$: request action where the second correlation value is unknown; a (binder for a) correlation variable *id* is used instead

  a corresponding response action can be $response(charge, 1234, id)$; the (free) occurrence of the correlation variable *id* indicates the connection with the action where the variable is bound

SocL actions

Actions ($a \in Act$)

have the form $t(i, c)$

- t: type of the action (e.g. *request*, *response*, *fail*, ...)
- i: name of the interaction which the action is part of (e.g. *charge*)
- c: tuple of correlation values and variables identifying the interaction; $\underline{var}$ denotes a binding occurrence of the correlation variable *var*

Examples

- $request(charge, 1234, 1)$: action starting an (instance of the) interaction *charge* which will be identified through the correlation tuple $\langle 1234, 1\rangle$

  a corresponding response action can be $response(charge, 1234, 1)$
- $request(charge, 1234, \underline{id})$: request action where the second correlation value is unknown; a (binder for a) correlation variable *id* is used instead

  a corresponding response action can be $response(charge, 1234, id)$; the (free) occurrence of the correlation variable *id* indicates the connection with the action where the variable is bound

SocL actions

Actions ($a \in Act$)

have the form $t(i, c)$

- t: type of the action (e.g. *request*, *response*, *fail*, ...)
- i: name of the interaction which the action is part of (e.g. *charge*)
- c: tuple of correlation values and variables identifying the interaction; $\underline{var}$ denotes a binding occurrence of the correlation variable *var*

Examples

- $request(charge, 1234, 1)$: action starting an (instance of the) interaction *charge* which will be identified through the correlation tuple $\langle 1234, 1\rangle$

  a corresponding response action can be $response(charge, 1234, 1)$
- $request(charge, 1234, \underline{id})$: request action where the second correlation value is unknown; a (binder for a) correlation variable *id* is used instead

  a corresponding response action can be $response(charge, 1234, id)$; the (free) occurrence of the correlation variable *id* indicates the connection with the action where the variable is bound

SocL atomic propositions

Atomic propositions ($\pi \in AP$)

have the form $p(i, c)$

- p: name of the proposition (*accepting_request*, *accepting_cancel*, ...)
- i: name of the interaction (e.g. *charge*)
- c: tuple of correlation values and free variables

Examples

- *accepting_request*(*charge*): proposition indicating that a state can accept requests for the interaction *charge* (regardless of the correlation data)
- *accepting_cancel*(*charge*, 1234, 1): a state permits to cancel those requests for interaction *charge* identified by the correlation tuple $\langle 1234, 1 \rangle$

SocL atomic propositions

Atomic propositions ($\pi \in AP$)

have the form $p(i, c)$

- p: name of the proposition (*accepting_request*, *accepting_cancel*, ...)
- i: name of the interaction (e.g. *charge*)
- c: tuple of correlation values and free variables

Examples

- $accepting\_request(charge)$: proposition indicating that a state can accept requests for the interaction *charge* (regardless of the correlation data)
- $accepting\_cancel(charge, 1234, 1)$: a state permits to cancel those requests for interaction *charge* identified by the correlation tuple $\langle 1234, 1\rangle$

SocL syntax

State formulae syntax

$$\phi \;::=\; \mathit{true} \;\mid\; \pi \;\mid\; \neg\phi \;\mid\; \phi \wedge \phi' \;\mid\; E\Psi \;\mid\; A\Psi$$

Path formulae syntax

$$\Psi \;::=\; X_\gamma \phi \;\mid\; \phi\,_\chi U_\gamma\,\phi' \;\mid\; \phi\,_\chi W_\gamma\,\phi'$$

Action formulae syntax

$$\gamma \;::=\; \underline{a} \;\mid\; \chi \qquad\qquad \chi \;::=\; tt \;\mid\; a \;\mid\; \tau \;\mid\; \neg\chi \;\mid\; \chi \wedge \chi$$

$\underline{a}$ indicates that the action may contain variables binders

Some derived modalities

$<\gamma>\phi$	stands for	$EX_\gamma\,\phi$	$[\gamma]\,\phi$	stands for	$\neg<\gamma>\neg\phi$
$E(\phi\,_\chi U\,\phi')$	stands for	$\phi' \vee E(\phi_\chi U_{\chi\vee\tau}\,\phi')$	$EF\phi$	stands for	$E(\mathit{true}\,_{tt}U\phi)$
$AF_\gamma\,\mathit{true}$	stands for	$A(\mathit{true}\,_{tt}U_\gamma\,\mathit{true})$	$AG\,\phi$	stands for	$\neg\,EF\,\neg\,\phi$

SocL syntax

State formulae syntax

$$\phi \quad ::= \quad \textit{true} \quad | \quad \pi \quad | \quad \neg\phi \quad | \quad \phi \wedge \phi' \quad | \quad E\Psi \quad | \quad A\Psi$$

E and *A* are existential and universal (resp.) *path quantifiers*

Action formulae syntax

$\gamma \quad ::= \quad \underline{a} \quad | \quad \chi \qquad\qquad \chi \quad ::= \quad tt \quad | \quad a \quad | \quad \tau \quad | \quad \neg\chi \quad | \quad \chi \wedge \chi$

$\underline{a}$ indicates that the action may contain variables binders

Some derived modalities

$< \gamma > \phi$	stands for	$EX_\gamma\, \phi$	$[\gamma]\,\phi$	stands for	$\neg < \gamma > \neg\phi$
$E(\phi\,_\chi U\,\phi')$	stands for	$\phi' \vee E(\phi_\chi U_{\chi\vee\tau}\, \phi')$	$EF\phi$	stands for	$E(\textit{true}\,_{tt}U\phi)$
$AF_\gamma\, \textit{true}$	stands for	$A(\textit{true}\,_{tt}U_\gamma\, \textit{true})$	$AG\,\phi$	stands for	$\neg\, EF \neg\,\phi$

SocL syntax

State formulae syntax

$$\phi \quad ::= \quad \mathit{true} \quad | \quad \pi \quad | \quad \neg\phi \quad | \quad \phi \wedge \phi' \quad | \quad E\Psi \quad | \quad A\Psi$$

Path formulae syntax

$$\Psi \quad ::= \quad X_\gamma \phi \quad | \quad \phi\ {}_\chi U_\gamma\ \phi' \quad | \quad \phi\ {}_\chi W_\gamma\ \phi'$$

Action formulae syntax

$$\gamma \quad ::= \quad \underline{a} \quad | \quad \chi \qquad\qquad \chi \quad ::= \quad tt \quad | \quad a \quad | \quad \tau \quad | \quad \neg\chi \quad | \quad \chi \wedge \chi$$

$\underline{a}$ indicates that the action may contain variables binders

Some derived modalities

$< \gamma > \phi$	stands for	$EX_\gamma\, \phi$	$[\gamma]\,\phi$	stands for	$\neg < \gamma > \neg\,\phi$
$E(\phi\ {}_\chi U\ \phi')$	stands for	$\phi' \vee E(\phi_\chi U_{\chi\vee\tau}\, \phi')$	$EF\phi$	stands for	$E(\mathit{true}\ {}_{tt} U \phi)$
$AF_\gamma\ \mathit{true}$	stands for	$A(\mathit{true}\ {}_{tt} U_\gamma\, \mathit{true})$	$AG\,\phi$	stands for	$\neg\, EF \neg\,\phi$

SocL syntax

State formulae syntax

$$\phi \ ::= \ \mathit{true} \ | \ \pi \ | \ \neg\phi \ | \ \phi \wedge \phi' \ | \ E\Psi \ | \ A\Psi$$

Path formulae syntax

$$\Psi \ ::= \ X_\gamma \phi \ | \ \phi \,{}_\chi U_\gamma\, \phi' \ | \ \phi \,{}_\chi W_\gamma\, \phi'$$

X, U and W are the *next*, *(strong) until* and *weak until* operators

- $X_\gamma \phi$ says that in the next state of the path, reached by an action satisfying γ, the formula ϕ holds
- $\phi \,{}_\chi U_\gamma\, \phi'$ says that ϕ' holds at some future state of the path reached by a last action satisfying γ, while ϕ holds from the current state until that state is reached and all the actions executed in the meanwhile along the path satisfy χ
- $\phi \,{}_\chi W_\gamma\, \phi'$ holds on a path either if the corresponding strong until operator holds or if for all the states of the path the formula ϕ holds and all the actions of the path satisfy χ

SocL syntax

State formulae syntax

$$\phi \ ::= \ true \mid \pi \mid \neg\phi \mid \phi \wedge \phi' \mid E\Psi \mid A\Psi$$

Path formulae syntax

$$\Psi \ ::= \ X_\gamma \phi \mid \phi\ {}_\chi U_\gamma\ \phi' \mid \phi\ {}_\chi W_\gamma\ \phi'$$

Action formulae syntax

$$\gamma \ ::= \ \underline{a} \mid \chi \qquad\qquad \chi \ ::= \ tt \mid a \mid \tau \mid \neg\chi \mid \chi \wedge \chi$$

$\underline{a}$ indicates that the action may contain variables binders

Some derived modalities

$<\gamma>\phi$ stands for $EX_\gamma\ \phi$		$[\gamma]\,\phi$ stands for $\neg<\gamma>\neg\phi$	
$E(\phi\ {}_\chi U\ \phi')$ stands for $\phi' \vee E(\phi_\chi U_{\chi\vee\tau}\ \phi')$		$EF\phi$ stands for $E(true\ {}_{tt}U\phi)$	
$AF_\gamma\ true$ stands for $A(true\ {}_{tt}U_\gamma\ true)$		$AG\,\phi$ stands for $\neg EF\neg\phi$	

SocL syntax

- $<\gamma>\phi$ states that it is *possible* to perform an action satisfying γ and thereby reaching a state that satisfies formula ϕ
- $[\gamma]\,\phi$ states that no matter how a process performs an action satisfying γ, the state it reaches in doing so will *necessarily* satisfy the formula ϕ
- $EF\phi$ means that there is some path that leads to a state at which ϕ holds; that is, ϕ *eventually* holds on some path
- $AF_\gamma\,\phi$ means that an action satisfying γ will be performed in the future along every path and at the reached states ϕ holds; if ϕ is *true*, we say that an action satisfying γ will *always eventually* be performed
- $AG\,\phi$ states that ϕ holds at every state on every path; that is, ϕ holds *globally*

Some derived modalities

$<\gamma>\phi$	stands for	$EX_\gamma\,\phi$	$[\gamma]\,\phi$	stands for	$\neg <\gamma> \neg\phi$
$E(\phi\,{}_\chi U\,\phi')$	stands for	$\phi' \vee E(\phi_\chi U_{\chi\vee\tau}\,\phi')$	$EF\phi$	stands for	$E(true\;{}_{tt}U\phi)$
$AF_\gamma\;true$	stands for	$A(true\;{}_{tt}U_\gamma\,true)$	$AG\,\phi$	stands for	$\neg\,EF\,\neg\,\phi$

SocL description of abstract properties

Availability

the service is always capable to accept a request

$$AG(accepting\_request(i))$$

Reliability

the service guarantees a successful response to each received request

$$AG[request(i, \underline{v})]AF_{response(i,v)}\ true$$

Responsiveness

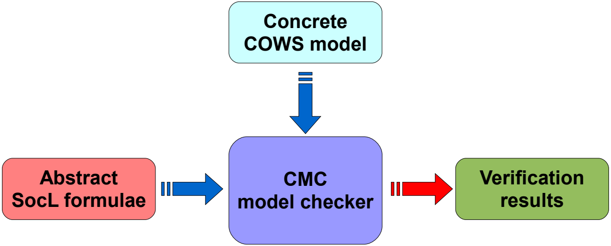
the service guarantees a response to each received request

$$AG[request(i, \underline{v})]\ AF_{response(i,v) \vee fail(i,v)}\ true$$

...

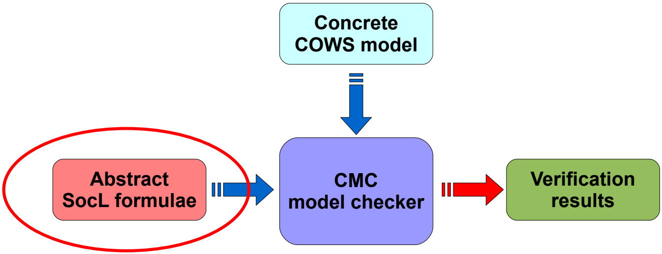
A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



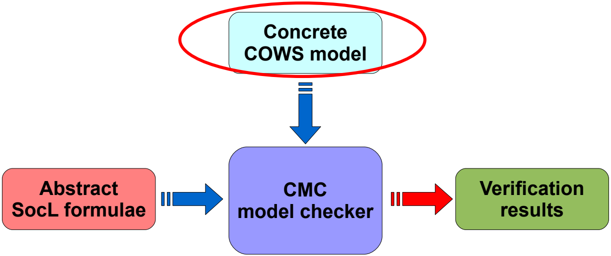
A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms

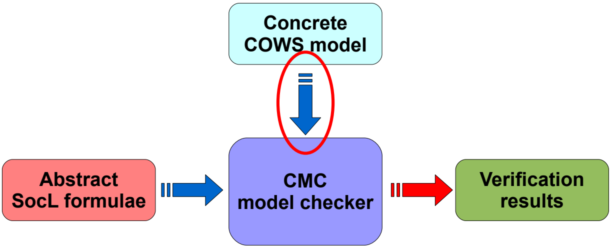
We resort to a linguistic formalism rather than directly using L^2TSs because

- L^2TSs are too low level
- L^2TSs suffer for lack of compositionality,

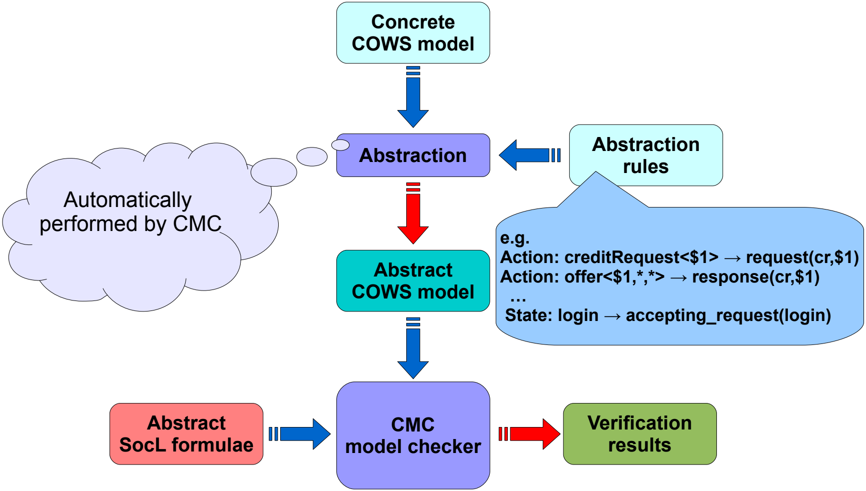
  i.e. they offer no means for constructing the L^2TS of a composed service in terms of the L^2TSs of its components
- linguistic terms are more intuitive and concise notations
- using linguistic terms, services are built in a compositional way
- linguistic terms are syntactically finite, even when the corresponding semantic model (i.e. L^2TSs) is not

A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place

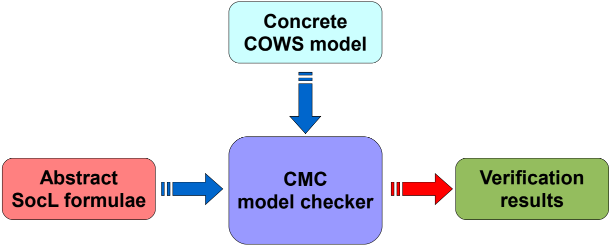


A novel verification methodology of service properties



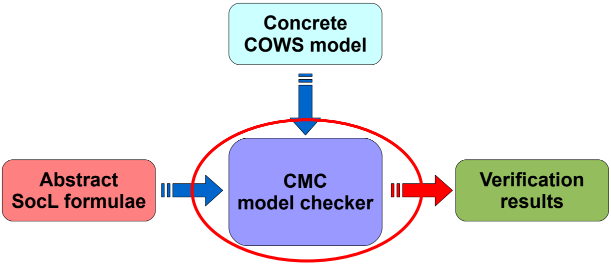
A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



The model checker CMC

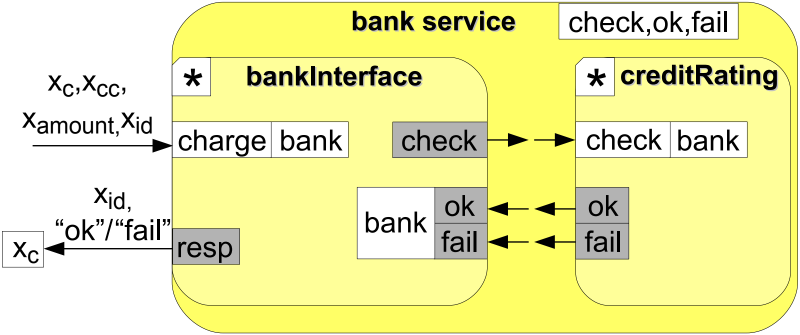
To assist the verification process of SocL formulae over L^2TS

- CMC is an efficient on-the-fly model checker
- The basic idea behind CMC is that, given a state of an L^2TS, the validity of a SocL formula on that state can be established by:
  - checking the satisfiability of the state predicates
  - analyzing the transitions allowed in that state
  - establishing the validity of some subformula in some/all of the next reachable states
- If a SocL formula is not satisfied, a *counterexample* is exhibited

CMC can be used to verify properties of services specified in COWS

CMC can be downloaded or experimented via its web interface at `http://fmt.isti.cnr.it/cmc`

Model checking the bank service



Model checking the bank service

The instantiation of the generic patterns of formulae over the bank service is obtained by just replacing any occurrence of *i* with *charge*

The bank service is *always available*

$$AG\,(accepting\_request(charge))$$

In every state the service may accept a request for the interaction *charge*

The bank service is *responsive*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v) \vee fail(charge,v)}\; true$$

The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

The bank service is *reliable*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v)}\; true$$

The service guarantees a successful response to each received request

Model checking the bank service

The instantiation of the generic patterns of formulae over the bank service is obtained by just replacing any occurrence of *i* with *charge*

The bank service is *always available*

$$AG\,(accepting\_request(charge))$$

In every state the service may accept a request for the interaction *charge*

The bank service is *responsive*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v) \vee fail(charge,v)}\ true$$

The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

The bank service is *reliable*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v)}\ true$$

The service guarantees a successful response to each received request

Model checking the bank service

The instantiation of the generic patterns of formulae over the bank service is obtained by just replacing any occurrence of *i* with *charge*

The bank service is *always available*

$$AG\,(accepting\_request(charge))$$

In every state the service may accept a request for the interaction *charge*

The bank service is *responsive*

$$AG\,[request(charge, \underline{v})]\ AF_{response(charge,v) \vee fail(charge,v)}\ true$$

The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

The bank service is *reliable*

$$AG\,[request(charge, \underline{v})]\ AF_{response(charge,v)}\ true$$

The service guarantees a successful response to each received request

Model checking the bank service

The instantiation of the generic patterns of formulae over the bank service is obtained by just replacing any occurrence of *i* with *charge*

The bank service is *always available*

$$AG\,(accepting\_request(charge))$$

In every state the service may accept a request for the interaction *charge*

The bank service is *responsive*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v) \vee fail(charge,v)}\ true$$

The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

The bank service is *reliable*

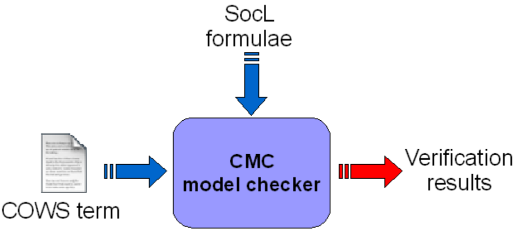
$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v)}\ true$$

The service guarantees a successful response to each received request

Tool demonstration . . .

Model checking: a calculus-based approach

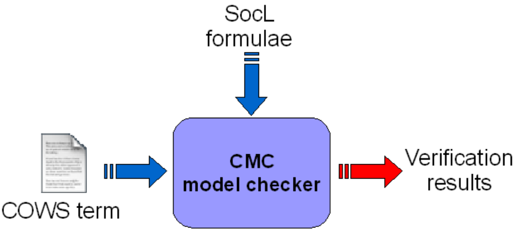
We have seen a calculus-based methodology for model checking COWS specifications



People in charge of verifying systems are required to understand and deal with calculi and logics.
This may not be the case, especially within , where people are usually familiar with higher level UML-based modelling languages.

Model checking: a calculus-based approach

We have seen a calculus-based methodology for model checking COWS specifications



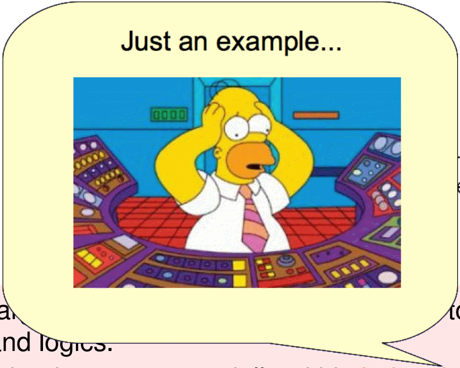
People in charge of verifying systems are required to understand and deal with calculi and logics.

This may not be the case, especially within industrial contexts, where people are usually familiar with higher-level UML-based modelling languages

Model checking: a calculus-based approach

We have seen a calculus-based methodology for model checking COWS specifications

ification
esults

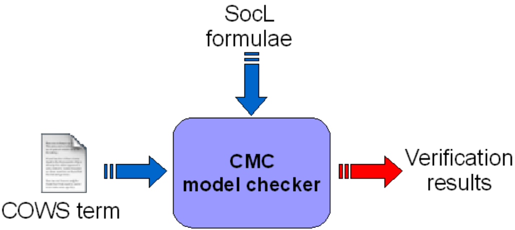


People in cha to understand and deal with calculi and logics.

This may not be the case, especially within industrial contexts, where people are usually familiar with higher-level UML-based modelling languages

Model checking: a calculus-based approach

We have seen a calculus-based methodology for model checking COWS specifications



People in charge of verifying systems are required to understand and deal with calculi and logics.
This may not be the case, especially within industrial contexts, where people are usually familiar with higher-level UML-based modelling languages

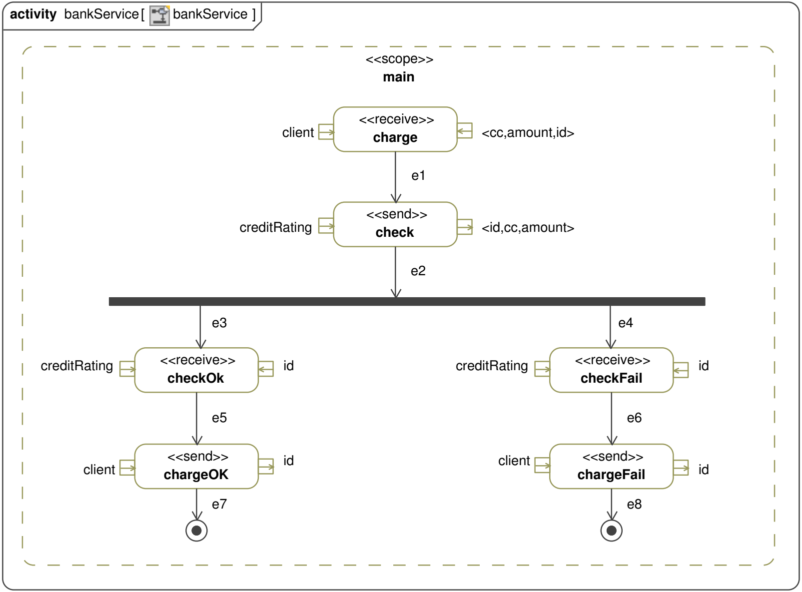
UML4SOA

- The most widely used language for modelling sw systems is UML
- UML4SOA is a UML 2.0 profile, inspired by WS-BPEL,
  that has been expressly designed for modeling service-oriented applications
- UML4SOA activity diagrams express the behavioral aspects of services
  - integrate UML with specialized actions for exchanging messages, specialized structured activity nodes and activity edges for representing scopes with event, fault and compensation handlers
- UML in general, and UML4SOA in particular, falls short of providing formal semantics and rigorous verification techniques
- Since UML4SOA specifications are static models, they are not suitable for direct automated analysis

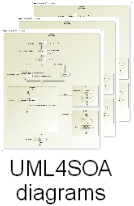
UML4SOA

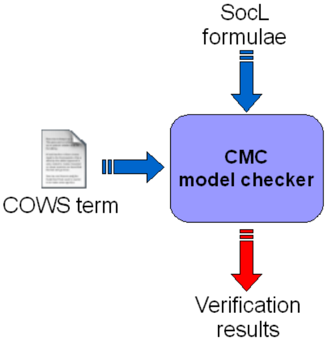
- The most widely used language for modelling sw systems is UML
- UML4SOA is a UML 2.0 profile, inspired by WS-BPEL, that has been expressly designed for modeling service-oriented applications
- UML4SOA activity diagrams express the behavioral aspects of services
  - integrate UML with specialized actions for exchanging messages, specialized structured activity nodes and activity edges for representing scopes with event, fault and compensation handlers
- UML in general, and UML4SOA in particular, falls short of providing formal semantics and rigorous verification techniques
- Since UML4SOA specifications are static models, they are not suitable for direct automated analysis

UML4SOA: diagram example

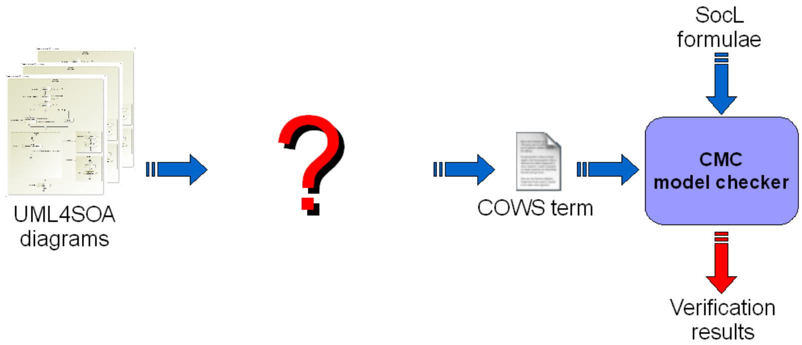


How to reconcile

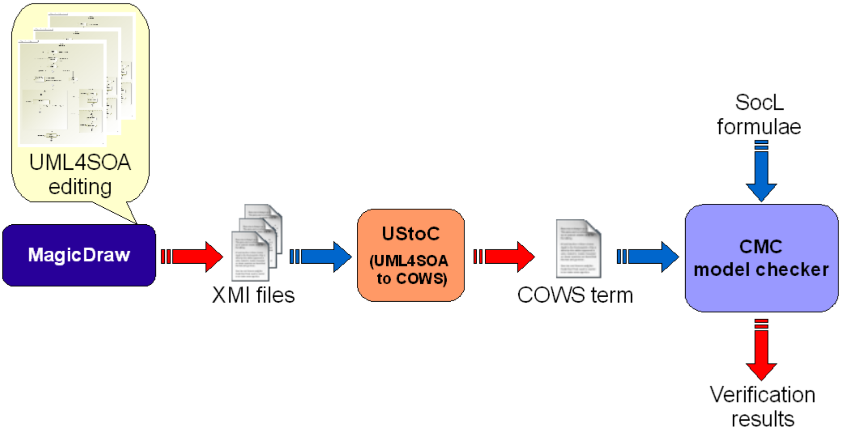




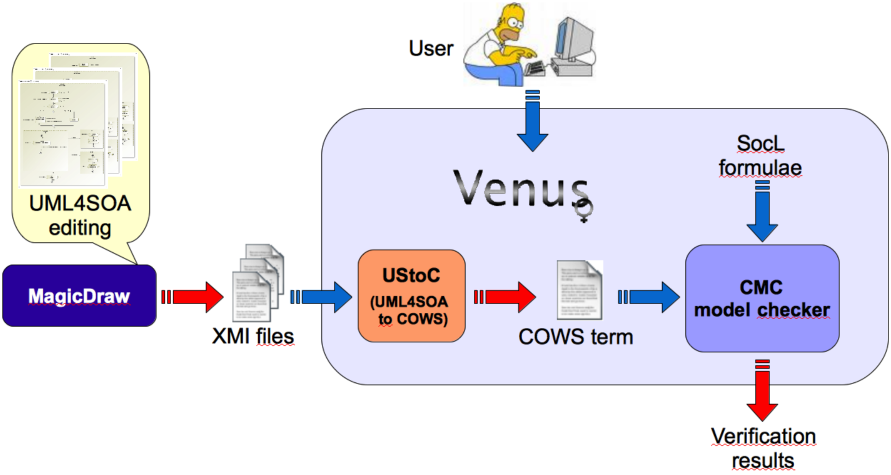
How to reconcile



Our proposal



Our proposal

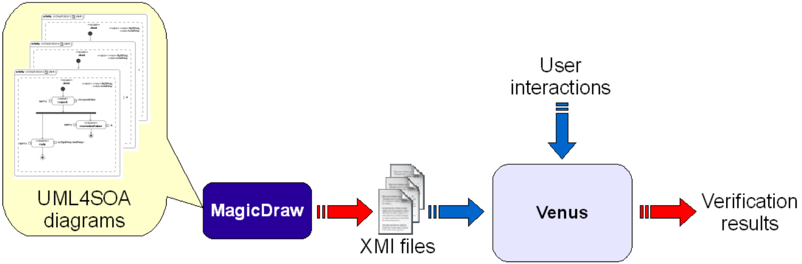


Our proposal

Venus: a Verification ENvironment for UML models of Services

A software environment for verifying behavioural properties of UML models of services by exploiting process calculi and temporal logics

- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation

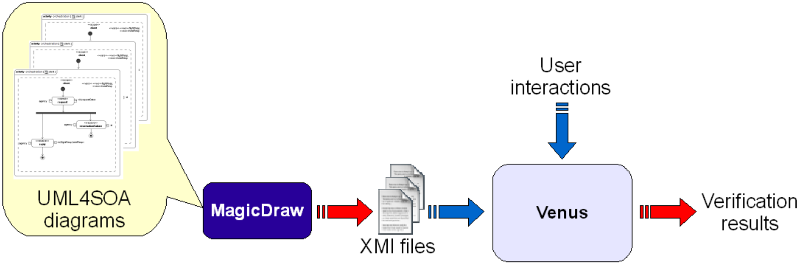


Our proposal

Venus: a Verification ENvironment for UML models of Services

A software environment for verifying behavioural properties of UML models of services by exploiting process calculi and temporal logics

- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation

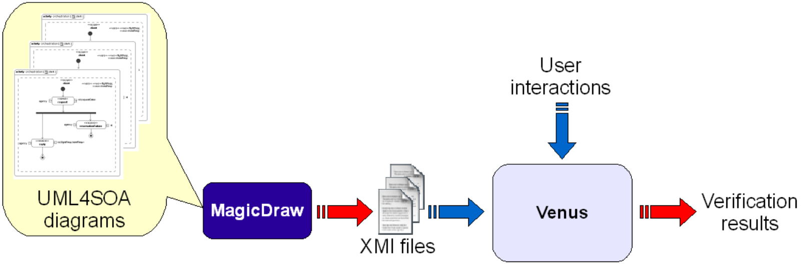


Our proposal

Venus: a Verification ENvironment for UML models of Services

A software environment for verifying behavioural properties of UML models of services by exploiting process calculi and temporal logics

- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation

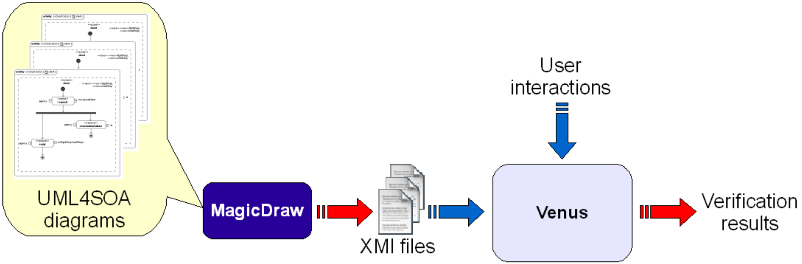


Our proposal

Venus: a **V**erification **EN**vironment for **U**ML models of **S**ervices

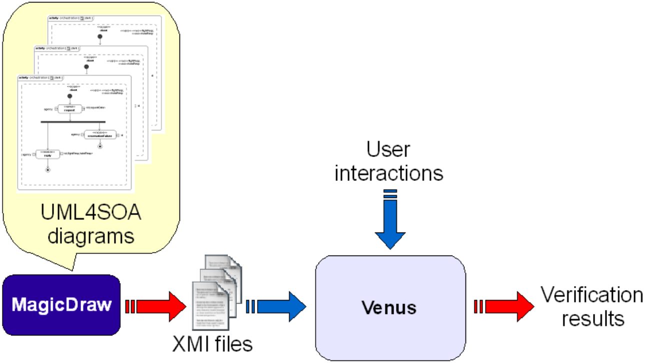
A software environment for verifying behavioural properties of UML models of services by exploiting process calculi and temporal logics

- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation

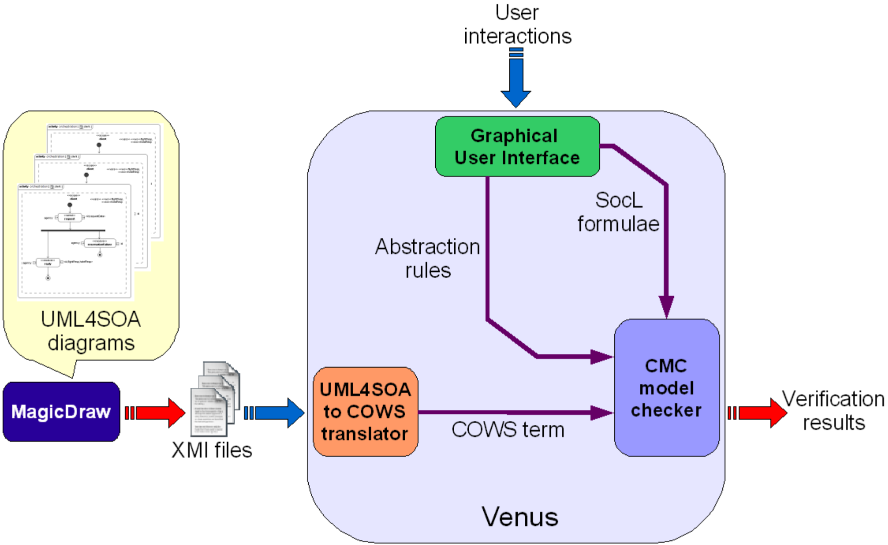


Tool demonstration . . .

Venus architecture



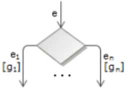
Venus architecture



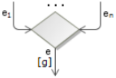
From UML4SOA to COWS

$creditRequest \bullet initialize?\langle x_{portal}, x_{id}, x_{name}, x_{pwd} \rangle$

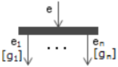
$x_{portal} \bullet initialize!\langle creditRequest, x_{id}, x_{userOk} \rangle$



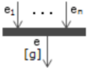
$$* \, e?\langle \mathbf{true} \rangle. \; [n_1, \dots, n_n] \, (\, n_1!\langle g_1 \rangle \mid \dots \mid n_n!\langle g_n \rangle \mid n_1?\langle \mathbf{true} \rangle. e_1!\langle \mathbf{true} \rangle \; + \; \dots \; + \; n_n?\langle \mathbf{true} \rangle. e_n!\langle \mathbf{true} \rangle \,)$$



$$* \, (\, e_1?\langle \mathbf{true} \rangle. e!\langle g \rangle \; + \dots + \; e_n?\langle \mathbf{true} \rangle. e!\langle g \rangle \,)$$



$$* \, e?\langle \mathbf{true} \rangle. \, (\, e_1!\langle g_1 \rangle \mid \dots \mid e_n!\langle g_n \rangle \,)$$



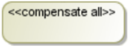
$$* \, e_1?\langle \mathbf{true} \rangle. \; \dots \; . e_n?\langle \mathbf{true} \rangle. e!\langle g \rangle$$

From UML4SOA to COWS

$$
\begin{array}{l}
[\mathtt{r}, \mathit{stack}] \\
(\, [k]\, (\mathrm{GRAPH}\ ;\ \{\!|\, c \bullet \mathit{main}?\langle\rangle.\, \mathrm{GRAPH_c} \,|\!\} \\
\qquad \mid \{\!|\, \mathit{Stack} \,|\!\} \mid *\, \mathrm{GRAPH_{ev}}\ ) \\
\ \mid \mathtt{r}?\langle\rangle.\, \{\!|\, \mathrm{GRAPH_e} \,|\!\}\ )
\end{array}
$$

$c \bullet \mathit{main}!\langle\rangle$

$\mathbf{kill}(k) \mid \{\!|\, \mathtt{r}!\langle\rangle \,|\!\}$



$\mathit{stack} \bullet \mathit{compAll}!\langle\rangle$

Our COWS implementation of UML4SOA constructs follows a compositional approach

From UML4SOA to COWS

$$
\begin{array}{l}
[\mathtt{r}, \mathit{stack}] \\
(\ [k]\ (\mathrm{GRAPH}\ ;\ \{\!|\ c \bullet \mathit{main}?\langle\rangle.\ \mathrm{GRAPH}_{c}\ |\!\} \\
\qquad\quad |\ \{\!|\ \mathit{Stack}\ |\!\}\ |\ *\ \mathrm{GRAPH}_{ev}\ ) \\
\quad |\ \mathtt{r}?\langle\rangle.\ \{\!|\ \mathrm{GRAPH}_{e}\ |\!\}\ )
\end{array}
$$

$c \bullet \mathit{main}!\langle\rangle$

$\mathbf{kill}(k)\ |\ \{\!|\ \mathtt{r}!\langle\rangle\ |\!\}$

$\mathit{stack} \bullet \mathit{compAll}!\langle\rangle$

Our COWS implementation of UML4SOA constructs follows a compositional approach

Concluding remarks

Conclusions

- COWS permits modelling different and typical aspects of services and Web services technologies
  - multiple start activities, receive conflicts, routing of correlated messages, service instances and interactions among them
- COWS can express the most common workflow patterns and can encode many other process and orchestration languages
- COWS, with some mild linguistic additions, can model all the relevant phases of the life cycle of service-oriented applications
  - publication, discovery, negotiation, deployment, orchestration, reconfiguration and execution

Conclusions

- Our observational semantics permits to check interchangeability of services and conformance against service specifications
- COWS type system permits specifying and forcing policies for constraining the services that can safely access any given datum
  - Types are just sets and operations on types are union, intersection, subset inclusion, ...
  - The runtime semantics only involves efficiently implementable operations on sets
- Our logical verification framework for checking functional properties of SOC applications has many advantages
  - It can be easily tailored to other service-oriented specification languages
  - SocL's parametric formulae permit expressing properties about many kinds of interaction patterns, e.g. *one-way*, *request-response*, *one request-multiple responses*, ...

On-going & future work

- Further analysis techniques
  - fully static variant of our type system
  - more powerful, *behavioural* type systems
  - an efficient symbolic characterisations of the labelled bisimilarities over a symbolic operational semantics
  - a formal account of COWS's expressiveness
  - analysis of security protocols for web service conversation, e.g. WS-SecureConversation and WS-Security

- Prototype implementations
  - a Java-based implementation based of COWS
  - an interpreter based on a symbolic operational semantics
  - a graphical editor (based on GMF) integrated with the interpreter

COWS

http://rap.dsi.unifi.it/cows/

Thank you!

Bibliography 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.

Bibliography 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.

Bibliography 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.

Bibliography 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