Towards Dynamic Updates in Service Composition

Mario Bravetti

Department of Computer Science
University of Bologna

INRIA research team FOCUS

Joint work with:
Gianluigi Zavattaro

Plan of the Talk

- Choreographies and Orchestrations
- Contract-based service discovery
- A dynamic update mechanism
- Conclusion

Web Service Choreography Description Language

- Describe the interaction among the combined services from a top abstract view

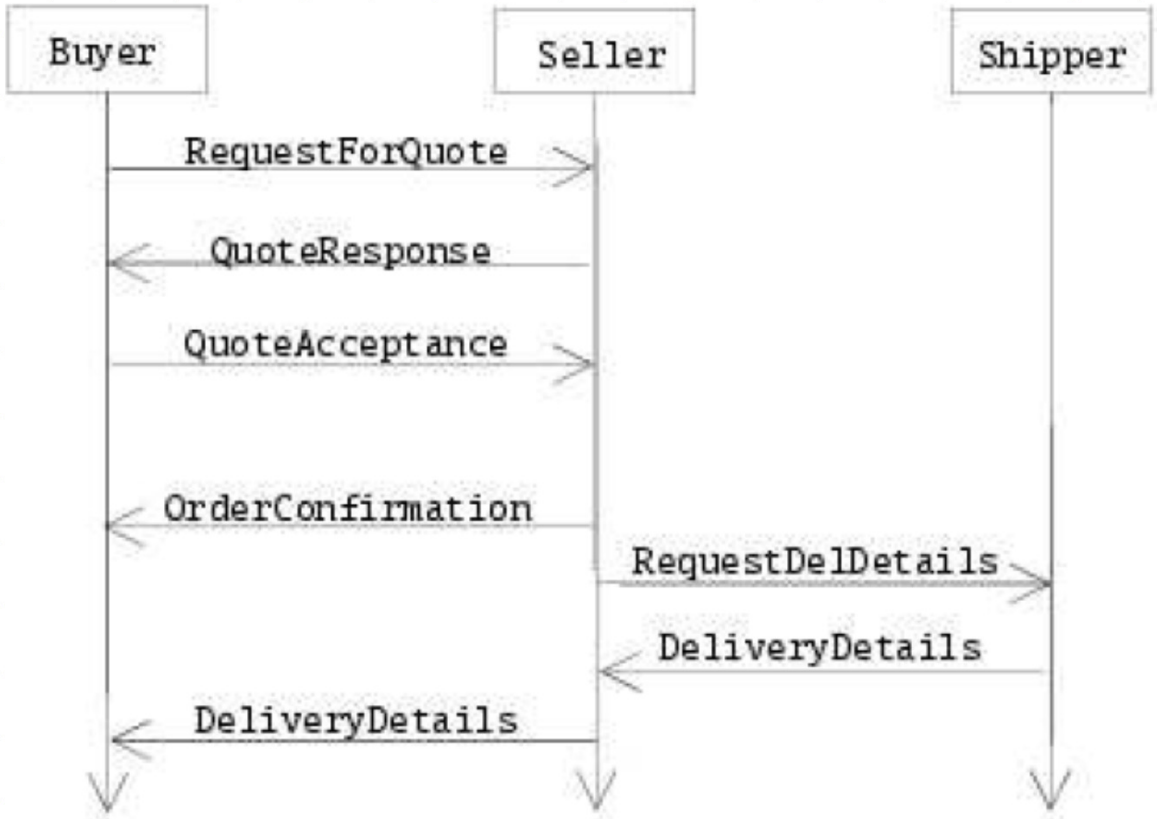
Choreography (e.g. WS-CDL)

Top abstract view of whole system: each action is a communication involving two of its participants

Orchestration (e.g. WS-BPEL)

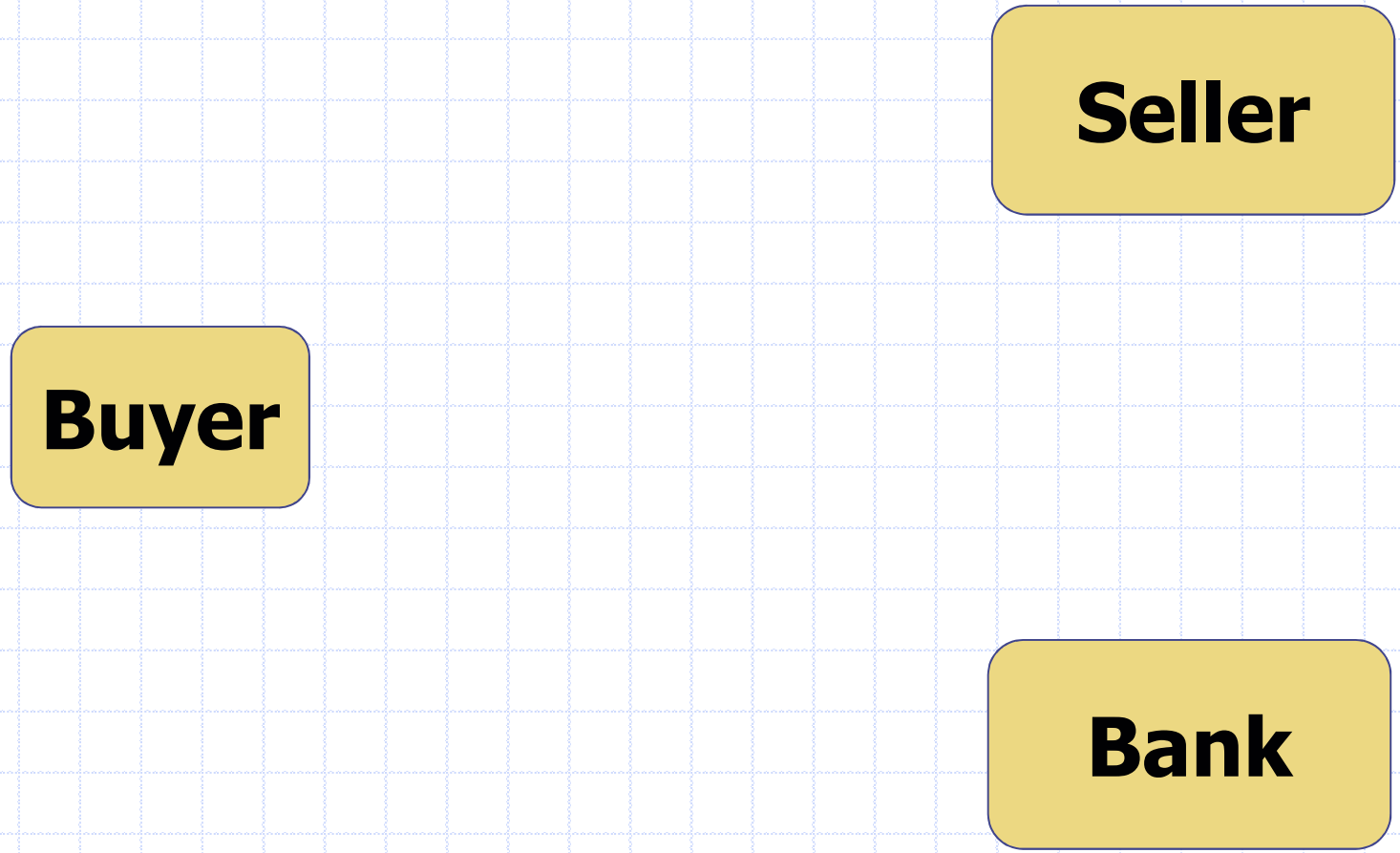
One Party detailed view of the system that orchestrates a part of it by sending (to other parties) & receiving messages

Similar to UML Sequence Diagrams



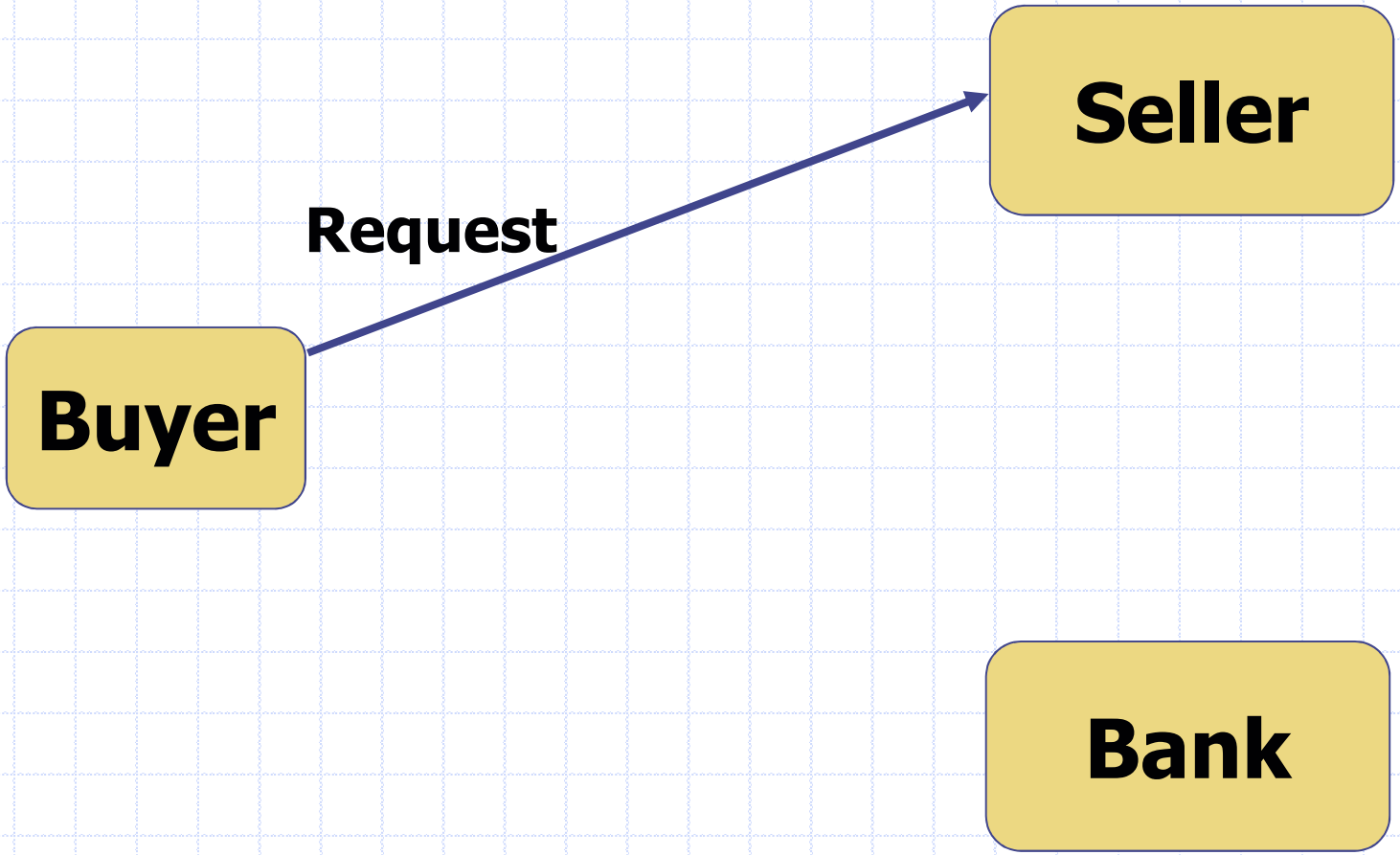
WS-CDL

- Global view of service interactions



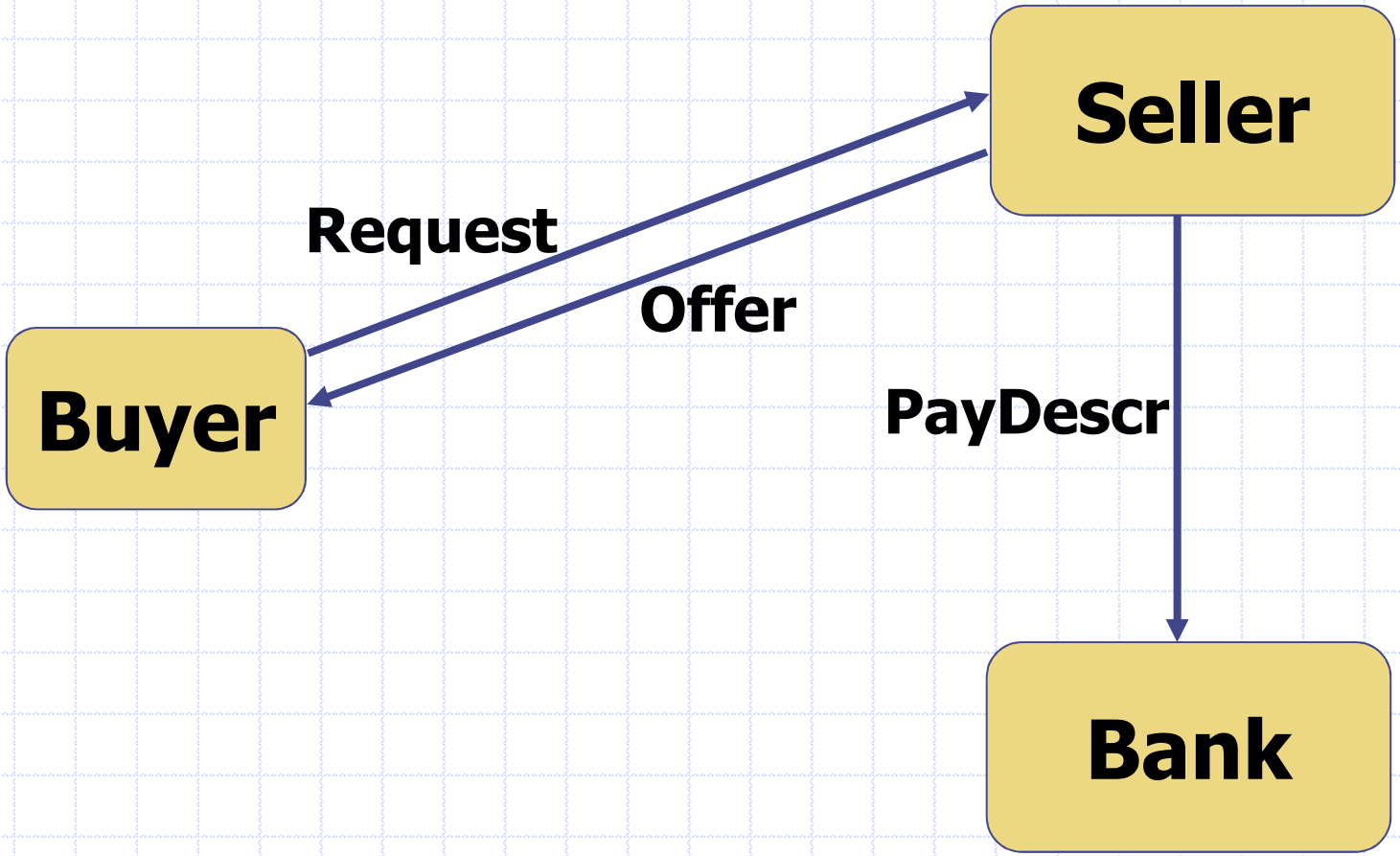
WS-CDL

- Global view of service interactions



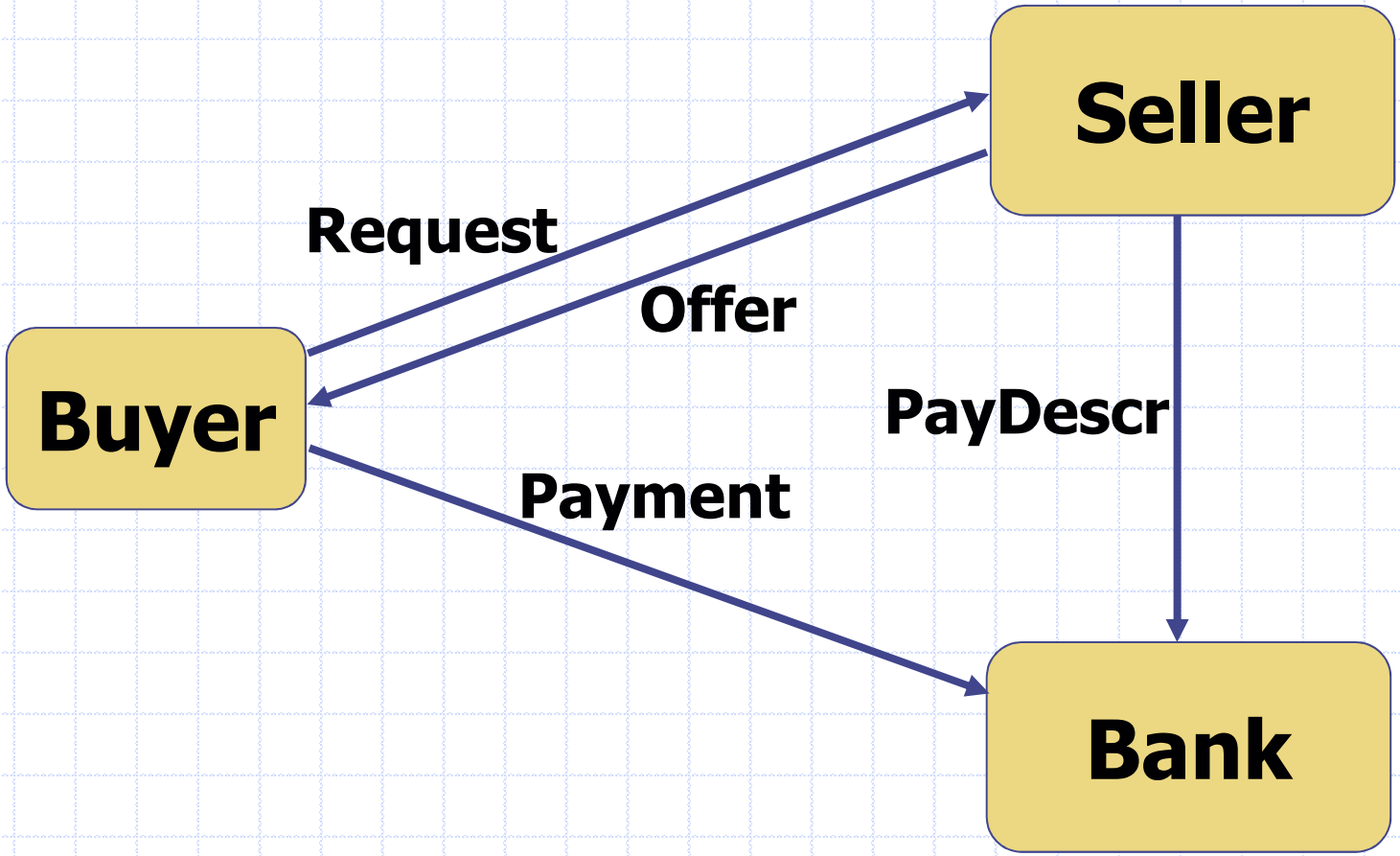
WS-CDL

- Global view of service interactions



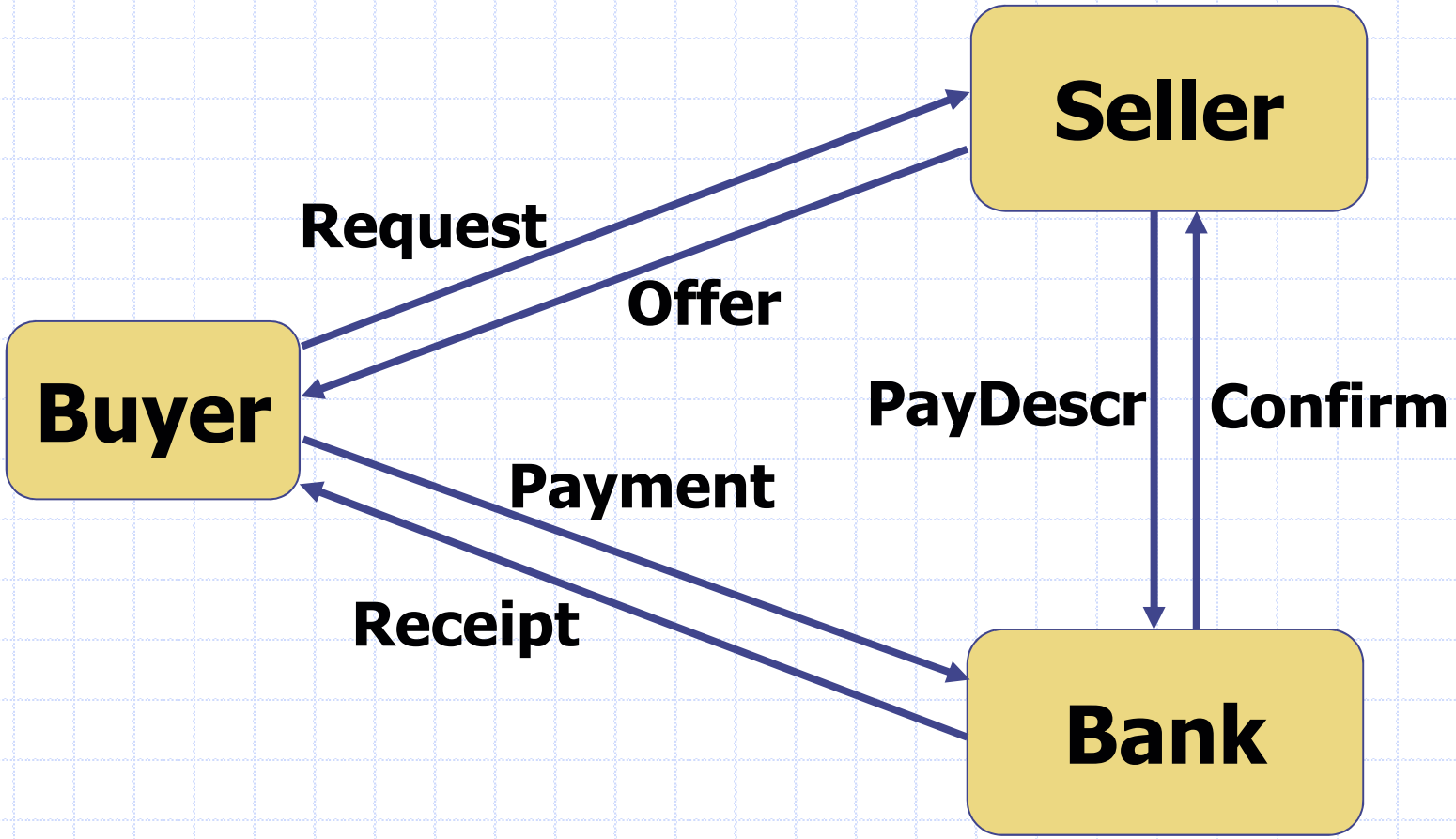
WS-CDL

- Global view of service interactions



WS-CDL

- Global view of service interactions



WS-CDL

$$\text{Request}_{\text{Buyer}\rightarrow\text{Seller}} \;;$$
$$(\; \text{Offer}_{\text{Seller}\rightarrow\text{Buyer}} \;|$$
$$\quad \text{PayDescr}_{\text{Seller}\rightarrow\text{Bank}} \;)\;;$$
$$\text{Payment}_{\text{Buyer}\rightarrow\text{Bank}} \;;$$
$$(\; \text{Confirm}_{\text{Bank}\rightarrow\text{Seller}} \;|$$
$$\quad \text{Receipt}_{\text{Bank}\rightarrow\text{Buyer}} \;)$$

Projection of the Choreography on the Single Participants

Buyer: Invoke(Request)@Seller;Receive(Offer);
Invoke(Payment)@Bank;Receive(Receipt)

Seller: Receive(Request);
(Invoke(Offer)@Buyer |
Invoke(PayDescr)@Bank);
Receive(Confirm)

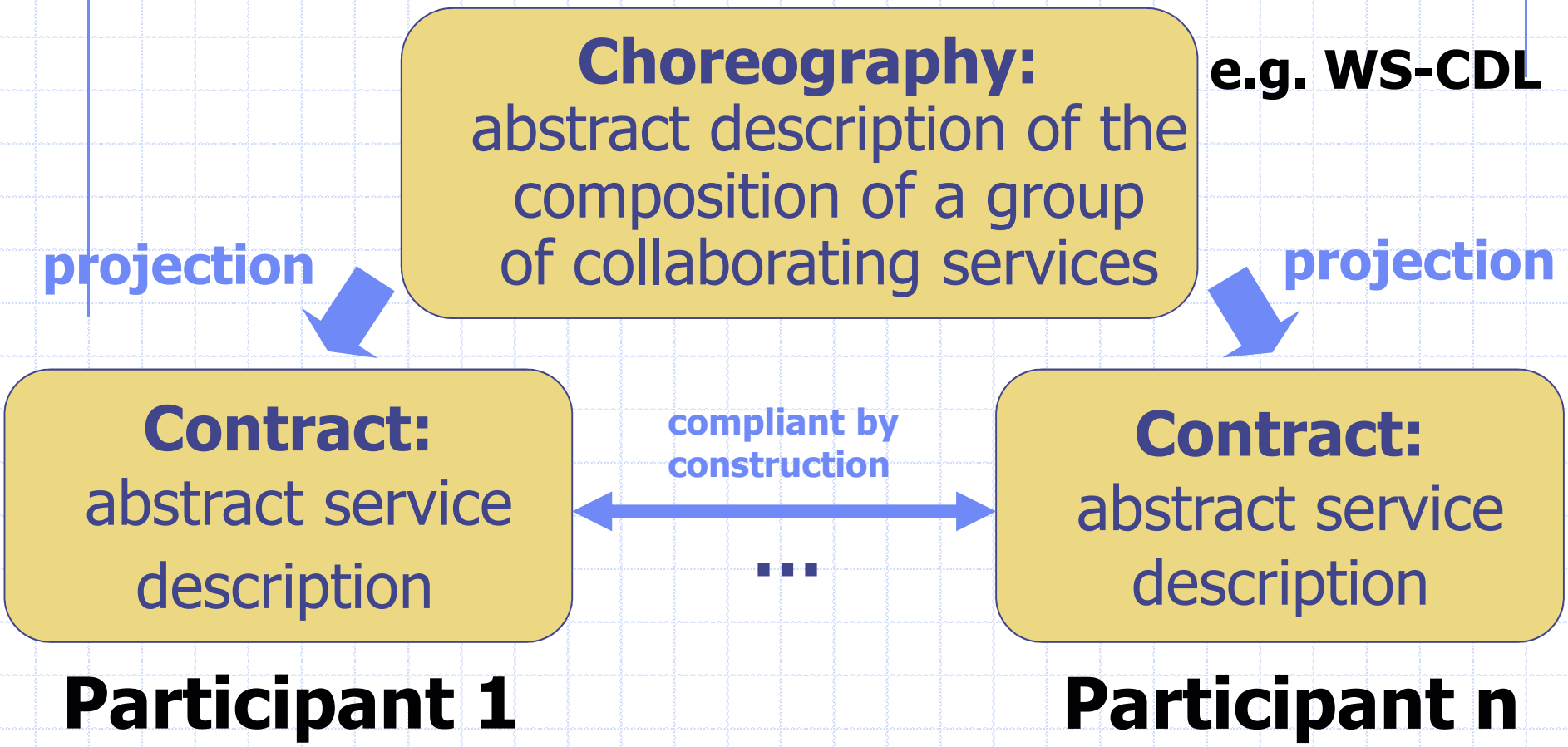
Bank: Receive(PayDescr);Receive(Payment);
(Invoke(Receipt)@Buyer |
Invoke(Confirm)@Seller)

Behavioural contracts and service retrieval

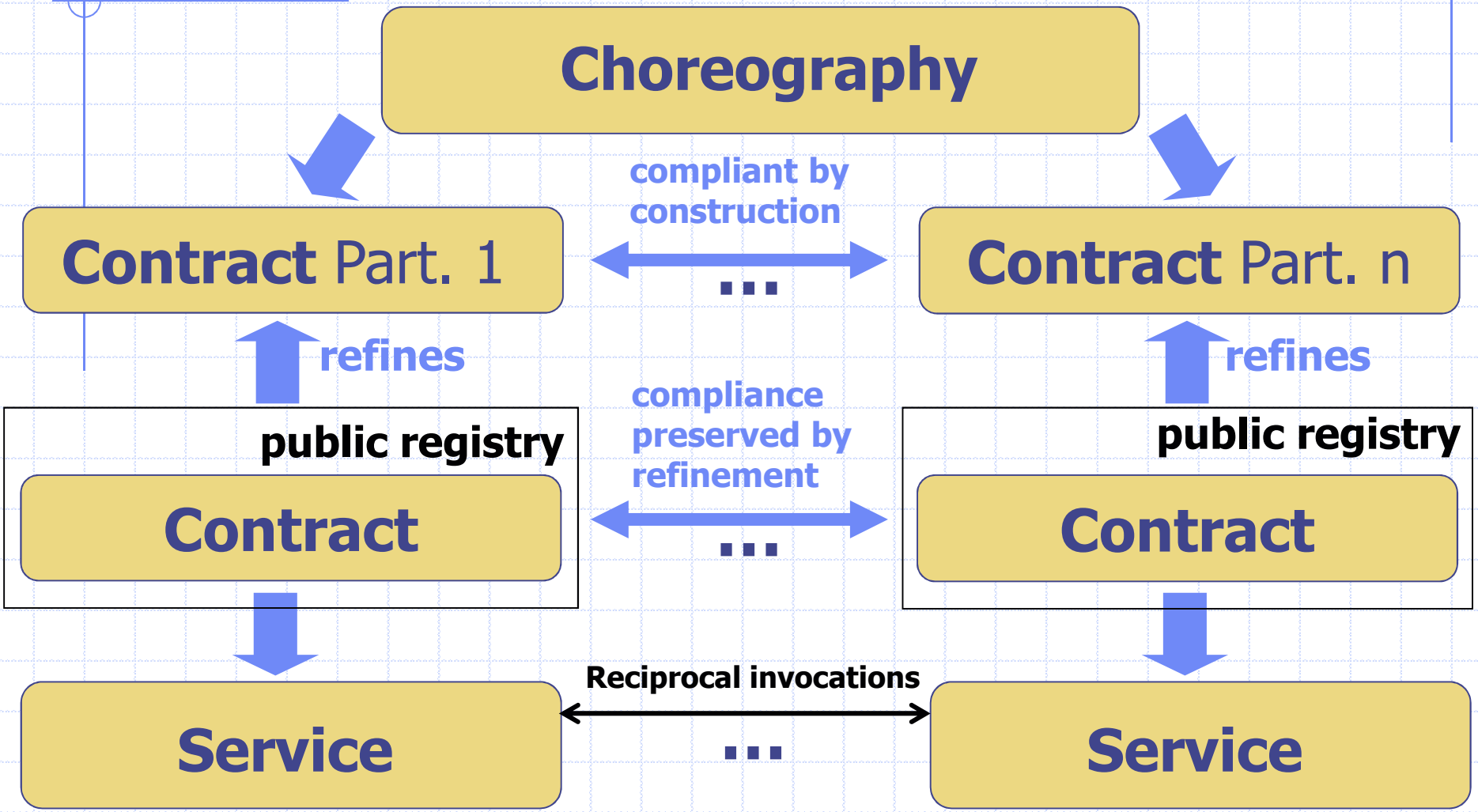
- Problem of retrieving in the internet services (by behavioral contracts) that:
  - interact without blocking (compliant)
  - can play the roles described by the choreography
- Useful notion: projection of choreography into contracts (one for each role)
- We cannot require that exactly projected contracts are retrieved

Deriving Set of Compliant Contracts from Choreography

[CHY07][BZ07b]



Compliance-Preserving Contract Refinement !



A Formal Model for WS-CDL

- A global choreography language:

$$H ::= a_{r \to s} \mid 1 \mid 0 \mid$$
$$H;H \mid H+H \mid H|H \mid H^*$$

A Formal Model for WS-CDL

- A global choreography language:

$$H ::= a_{r \to s} \mid 1 \mid 0 \mid H;H \mid H+H \mid H|H \mid H^*$$

r invokes the operation a of s

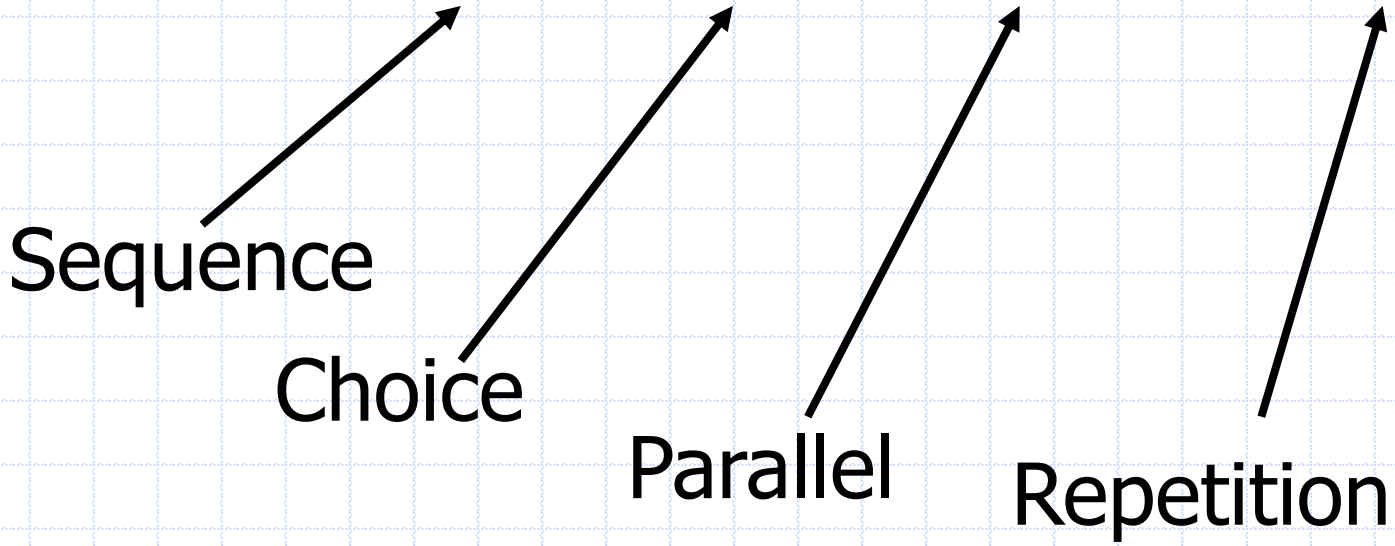
Successful termination

Unsuccessful termination

A Formal Model for WS-CDL

- A global choreography language:

$$H ::= a_{r \to s} \mid 1 \mid 0 \mid H;H \mid H+H \mid H|H \mid H^*$$



Standard semantics

- Standard semantics where:
  - $a_{r \to s}$ produces a $a_{r \to s}$ transition to 1
  - 1 produces a $\surd$ transition to 0

Semantics of choreographies

$$(\textsc{Comm})\ a_{r_1 \to r_2} \xrightarrow{a_{r_1 \to r_2}} \mathbf{1} \qquad (\textsc{One})\ \mathbf{1} \xrightarrow{\surd} \mathbf{0} \qquad (\textsc{Cho})\ \frac{H_1 \xrightarrow{\alpha} H_1'}{H_1 + H_2 \xrightarrow{\alpha} H_1'}$$

$$(\textsc{Seq})\ \frac{H_1 \xrightarrow{a_{r_1 \to r_2}} H_1'}{H_1;\ H_2 \xrightarrow{a_{r_1 \to r_2}} H_1';\ H_2} \qquad (\textsc{SeqTick})\ \frac{H_1 \xrightarrow{\surd} H_1' \quad H_2 \xrightarrow{\alpha} H_2'}{H_1;\ H_2 \xrightarrow{\alpha} H_2'}$$

$$(\textsc{Par})\ \frac{H_1 \xrightarrow{a_{r_1 \to r_2}} H_1'}{H_1 \mid H_2 \xrightarrow{a_{r_1 \to r_2}} H_1' \mid H_2} \qquad (\textsc{ParTick})\ \frac{H_1 \xrightarrow{\surd} H_1' \quad H_2 \xrightarrow{\surd} H_2'}{H_1 \mid H_2 \xrightarrow{\surd} H_1' \mid H_2'}$$

$$(\textsc{Star})\ \frac{H \xrightarrow{a_{r_1 \to r_2}} H'}{H^* \xrightarrow{a_{r_1 \to r_2}} H';\ H^*} \qquad (\textsc{StarTick})\ H^* \xrightarrow{\surd} \mathbf{0}$$

A Formal Model for orchestrations

- A language for orchestrations:

$$C ::= a \mid \overline{a}_r \mid \tau \mid 1 \mid 0 \mid C;C \mid C+C \mid C|C \mid C^*$$

$$P ::= [C]_r \mid P|P$$

A Formal Model for orchestrations

- A language for orchestrations:

$$C ::= a \mid \overline{a}_r \mid \tau \mid 1 \mid 0 \mid C;C \mid C+C \mid C|C \mid C^*$$

$$P ::= [C]_r \mid P|P$$

receive on a

invoke a at r

internal

Successful termination

Unsuccessful termination

A Formal Model for orchestrations

- A language for orchestrations:

$$C ::= a \mid \overline{a}_r \mid \tau \mid 1 \mid 0 \mid C;C \mid C+C \mid C|C \mid C^*$$

$$P ::= [C]_r \mid P|P$$

Sequence

Choice

Parallel

Repetition

A Formal Model for orchestrations

- A language for orchestrations:

$$C ::= a \mid \overline{a}_r \mid \tau \mid 1 \mid 0 \mid C;C \mid C+C \mid C|C \mid C^*$$

$$P ::= [C]_r \mid P|P$$

Behaviour of participant r

Parallel composition of participants

Standard semantics

- Standard semantics where:
  - $\overline{a}_s$ transition of role r synchronizes with a transition of role s and produces a $a_{r \to s}$ transition
  - 1 produces a $\surd$ transition to 0
  - $\surd$ is synchronized over parallel

Notion of Implementation

- Given a choreography $\mathbf{H}$, a system $\mathbf{P}$ implements $\mathbf{H}$ if:
  - $\mathbf{P}$ is a composition of compliant contracts
    - Intuitively they all always reach termination $\surd$ (we will see)
  - Each completed ($\surd$ terminating) weak (τ abstracting) trace of $\mathbf{P}$ is a completed trace of $\mathbf{H}$
    - all computations of $\mathbf{P}$ are correct conversations according to the choreography $\mathbf{H}$

Examples of choreography Implementations:

- Given the choreography:

$$\mathbf{Request}_{\mathbf{Alice}\rightarrow\mathbf{Bob}};\ (\mathbf{Accept}_{\mathbf{Bob}\rightarrow\mathbf{Alice}} + \mathbf{Reject}_{\mathbf{Bob}\rightarrow\mathbf{Alice}})$$

The following are implementations:

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}};(\mathbf{Accept}+\mathbf{Reject})]_{\mathbf{Alice}} \mid$$
$$[\mathbf{Request};(\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

Examples of choreography Implementations:

- Given the choreography:

$$\mathbf{Request}_{\mathbf{Alice}\rightarrow\mathbf{Bob}};\ (\mathbf{Accept}_{\mathbf{Bob}\rightarrow\mathbf{Alice}} + \mathbf{Reject}_{\mathbf{Bob}\rightarrow\mathbf{Alice}})$$

The following are implementations:

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}};(\mathbf{Accept}+\mathbf{Reject})]_{\mathbf{Alice}} \mid [\mathbf{Request};(\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}};(\mathbf{Accept}+\mathbf{Reject}+\mathbf{Retry})]_{\mathbf{Alice}} \mid [\mathbf{Request};(\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

Examples of choreography Implementations:

- Given the choreography:

$$\mathbf{Request}_{\mathbf{Alice}\rightarrow\mathbf{Bob}};\ (\mathbf{Accept}_{\mathbf{Bob}\rightarrow\mathbf{Alice}} + \mathbf{Reject}_{\mathbf{Bob}\rightarrow\mathbf{Alice}})$$

The following are implementations:

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}};(\mathbf{Accept}+\mathbf{Reject})]_{\mathbf{Alice}} \mid [\mathbf{Request};(\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}};(\mathbf{Accept}+\mathbf{Reject}+\mathbf{Retry})]_{\mathbf{Alice}} \mid [\mathbf{Request};(\overline{\mathbf{Accept}}_{\mathbf{Alice}}+\overline{\mathbf{Reject}}_{\mathbf{Alice}})]_{\mathbf{Bob}}$$

$$[\overline{\mathbf{Request}}_{\mathbf{Bob}};(\mathbf{Accept}+\mathbf{Reject}+\mathbf{Retry})]_{\mathbf{Alice}} \mid [\mathbf{Request};\overline{\mathbf{Accept}}_{\mathbf{Alice}}]_{\mathbf{Bob}}$$

"Canonical" Projection and Well-formed Choreography [SC 2007]

- Canonical projection $[\![\ ]\!]_t$:

$$[\![\ a_{r\to s}\ ]\!]_t \quad = \quad \begin{cases} \overline{a}_s & \text{if } t=r \\ a & \text{if } t=s \\ 1 & \text{otherwise} \end{cases}$$

$[\![H;H']\!]_t=[\![H]\!]_t\,;\,[\![H']\!]_t \qquad [\![H \mid H']\!]_t=[\![H]\!]_t \mid [\![H']\!]_t$

$[\![H+H']\!]_t=[\![H]\!]_t + [\![H']\!]_t \qquad [\![H^*]\!]_t=[\![H]\!]_t^*$

- H is well-formed if the system P, achieved via canonical projections, implements H

Example

- Consider the choreography H:

  $$a_{r \to s} ; b_{t \to u}$$

- Projection:

  $$[\, \overline{a}_s ; 1]_r \mid [\, a ; 1 \,]_s \mid [\, 1 ; \overline{b}_u \,]_t \mid [\, 1 ; b \,]_u$$

- Is H well-formed?
  - NO
  - But, if r=t…. YES

    $$[\, \overline{a}_s ; \overline{b}_u \,]_r \mid [\, a ; 1 \,]_s \mid [\, 1 ; b \,]_u$$

Connected Choreography (with Ivan Lanese)

- Sufficient conditions
  - unique point of choice,
  - connectedness of sequence,
  - causality safety

  that guarantee that H is well-formed
  - i.e. projection of a connected choreography H produces an implementation of H

Unique point of choice

- In a choice **H+H'**
  - The sender of the initial transitions in **H** and in **H'** is always the same
  - The roles in **H** and in **H'** are the same
- Example: if we drop the second condition

$$(a_{r\to s} + b_{r\to t})\,;\, c_{s\to t}$$

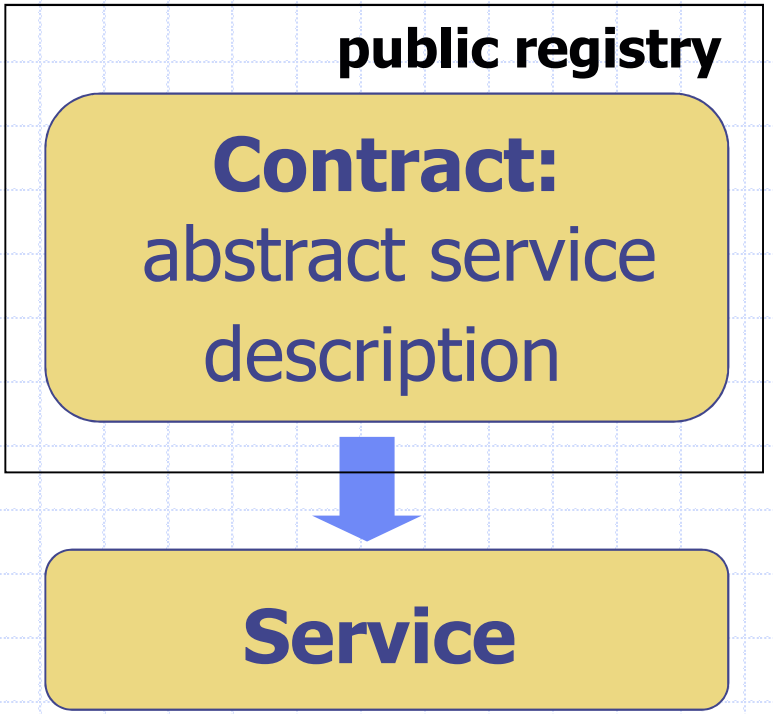
$$[\,(\overline{a}_s + \overline{b}_t)\,;1\,]_r \mid [\,(a+1)\,;\overline{c}_t\,]_s \mid [\,(1+b)\,;c\,]_t$$

Plan of the Talk

- Choreographies and Orchestrations
- Contract-based service discovery
- A dynamic update mechanism
- Conclusion

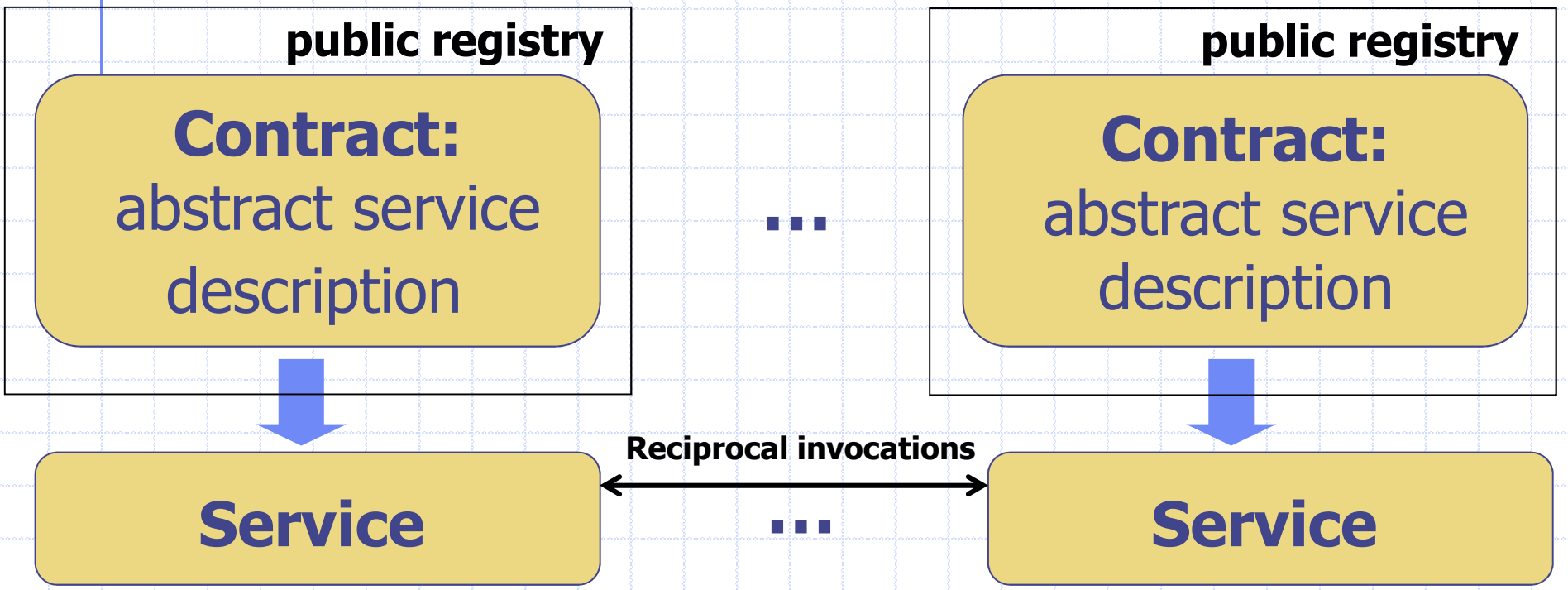
Contracts

- Contract: service "behavioural interface"
  - correct sequences of invoke and receive
  - as in an orchestration (role of a coreography)
  - just finite-state labeled transition systems with successful termination



Contract Compliance

- Verification of correctness of service composition based on their contracts: successful interaction i.e. no deadlock / termination reached



Service Compliance: Formally

- Services are compliant if the following holds for their composition $\mathbf{P}$:

  $\mathbf{P} \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} \mathbf{P}'$

  implies that there exists $\mathbf{P}''$ s.t.

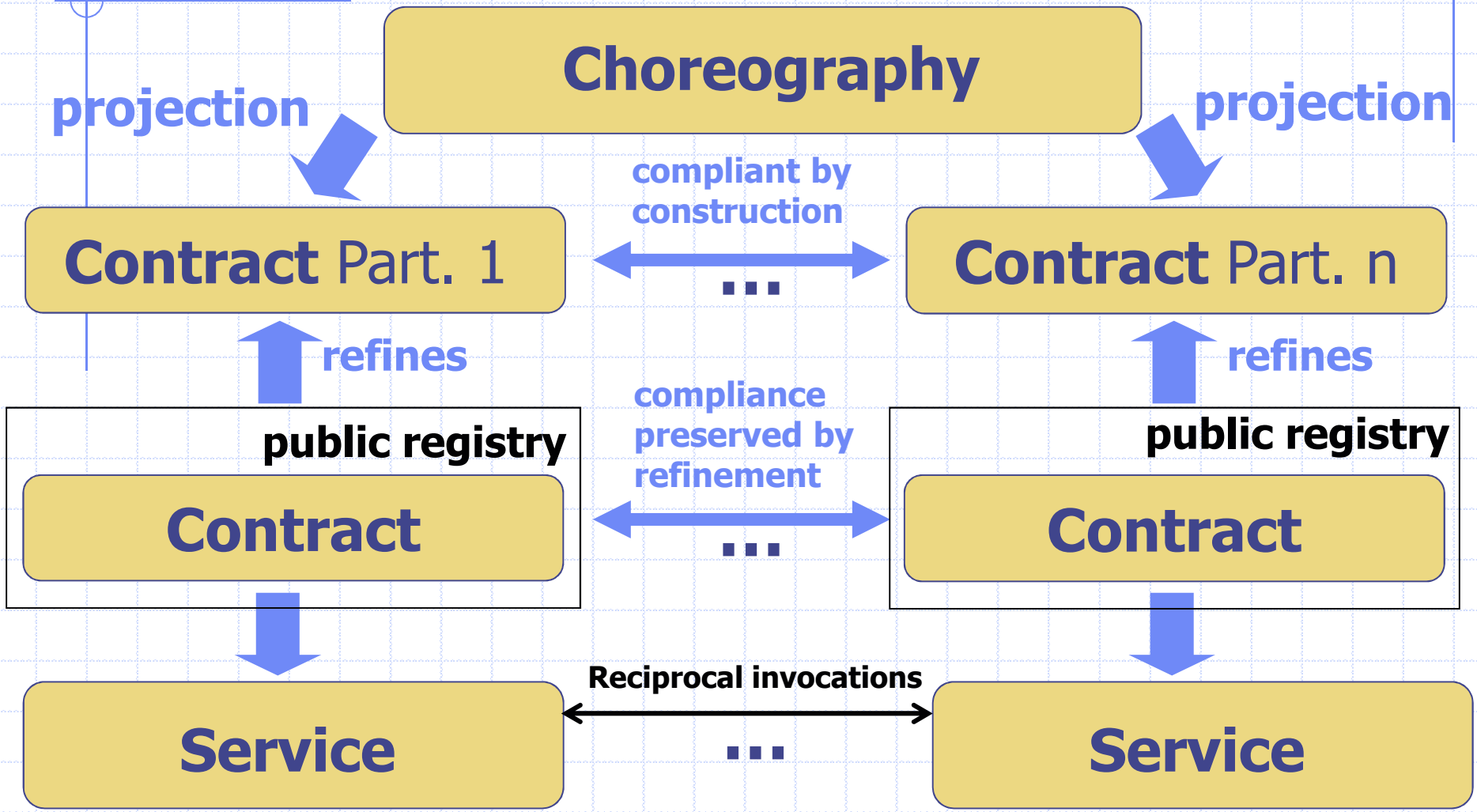
  $\mathbf{P}' \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_m} \mathbf{P}'' \xrightarrow{\surd}$

  - i.e. every computation can be extended to reach successful completion of all services
  - termination under fairness assumption

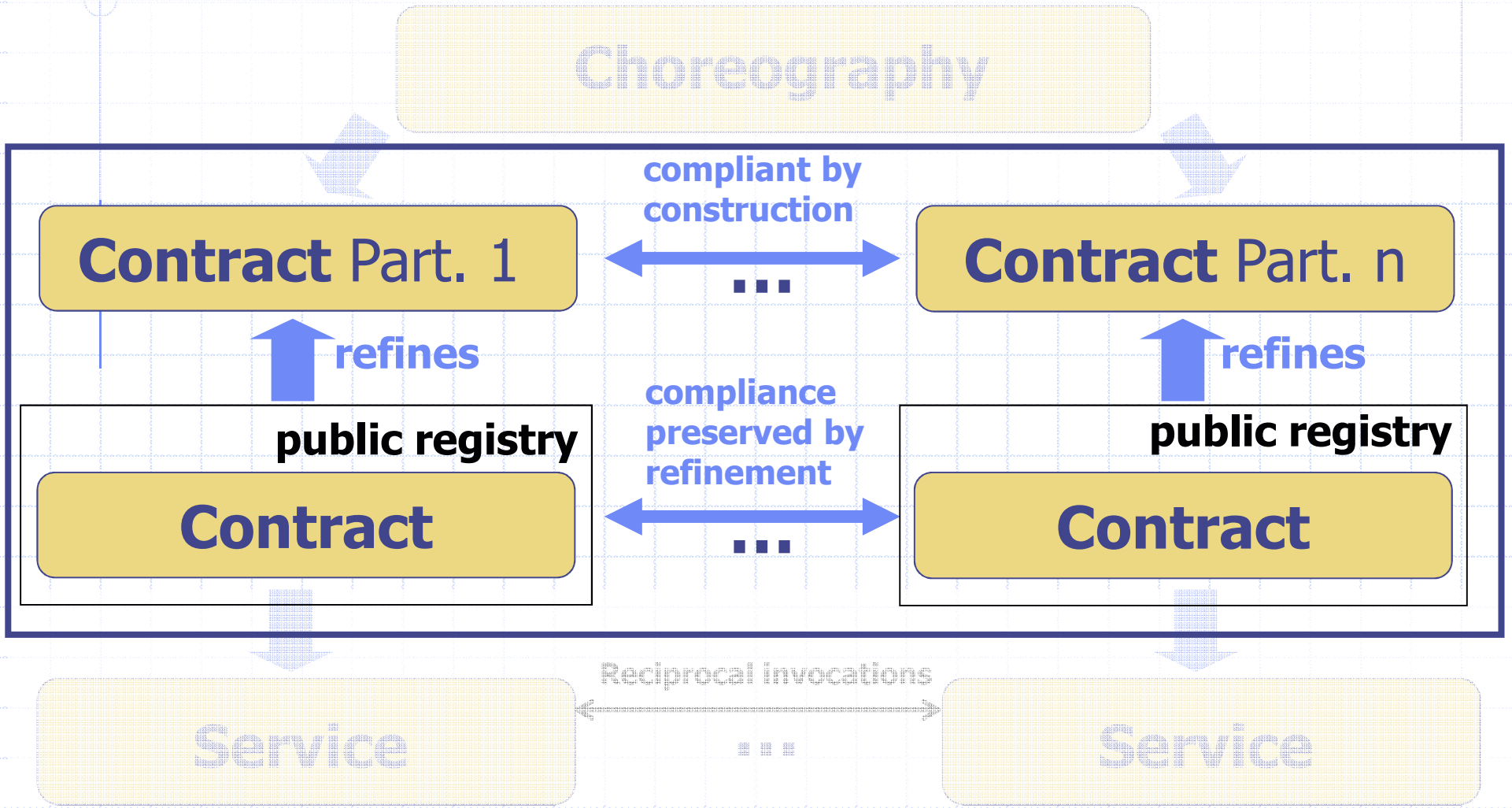
Example: compliant services

- The following pairs of services are compliant:
  - $C_1 = a+b+c$ $\quad C_2 = \overline{a} + \overline{b}$
  - $C_1 = a;b$ $\quad C_2 = \overline{a} \mid \overline{b}$
  - $C_1 = (a; \overline{b})^*$ $\quad C_2 = \overline{a};(b;\overline{a})^*;b$

Compliance-Preserving Contract Refinement !

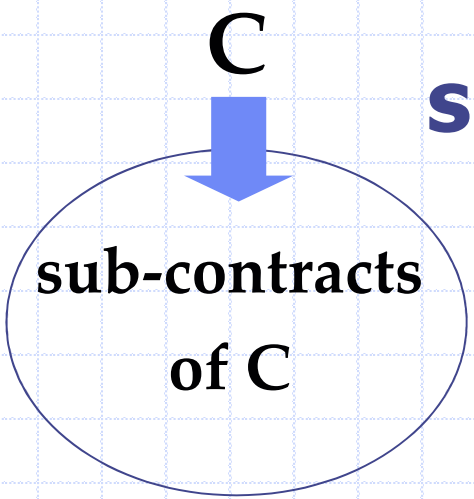


Contract Refinement Relation



Formally: Subcontract Preorder

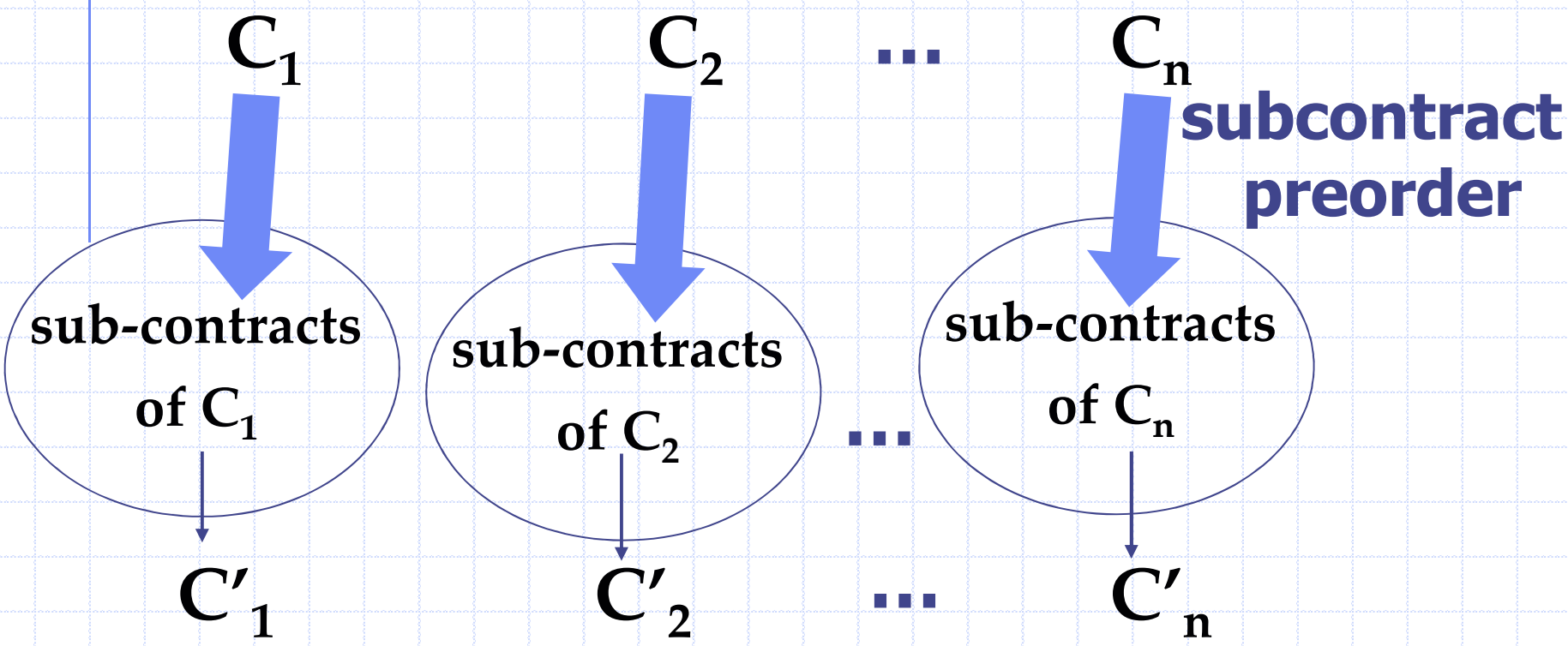
- Preorder ≤ between contracts C:
  - C′ ≤ C means C′ is a subcontract of C



subcontract preorder

Definition of Preorder Induced from Independent Refinement

Given a set of compliant contracts



is a set of compliant contracts

No maximal subcontract preorder … in general

- Consider the system:

  $[\, a \,] \mid [\, \overline{a} \,]$

  we could have one preorder $\leq_1$ for which

  $a + c.0 \leq_1 a$ $\qquad$ $\overline{a} + c.0 \leq_1 \overline{a}$

  and one preorder $\leq_2$ for which

  $a + \overline{c}.0 \leq_2 a$ $\qquad$ $\overline{a} + \overline{c}.0 \leq_2 \overline{a}$

  but no subcontract preorder could have

  $a + c.0 \leq a$ $\qquad$ $\overline{a} + \overline{c}.0 \leq \overline{a}$

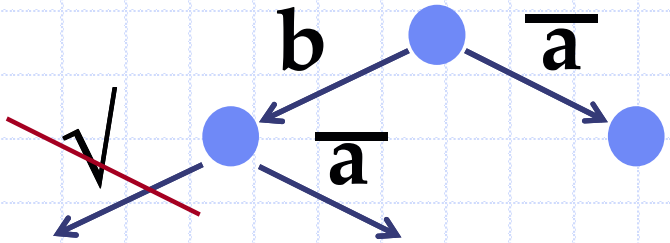
- Consequence: no independent refinement!

Maximal pre-order

- It exists changing some assumptions (asymmetry between inputs and outputs)
  - Constraining the structure of contracts:
    - outputs choosen internally (output persistence)
  - Strengthening the notion of compliance:
    - when an output is performed a corresponding input is required to be already enabled, like in ready-send of MPI (strong compliance)
  - Moving to asynchronous communication (e.g. via message queues)

Output persistence

- Output persistence means that given a process state **P**:
  - If **P** has an output transition on **a** and **P**-α->**P'** with α different from output on **a**, then also **P'** has an output transition on **a** (and **P'** -$\surd$-/->)



Syntactically… (guarantees output persistance)

- This holds, for instance, in WS-BPEL
  - Outputs cannot resolve the pick operator for external choices (the decision to execute outputs is taken internally)
- External choice among inputs: $a + b$ but

~~$\overline{a} + \overline{b}$~~ $\quad \tau;\overline{a} + \tau;\overline{b}$

~~$a + \overline{b}$~~ $\quad a + \tau;\overline{b}$

no external choice among outputs (and mixed choice) in contracts!

Choreography implementations (with output persistent contracts)

- Projection modified: $[\![a_{r\to s}]\!]_t = \tau;\overline{a_s}$ if $t=r$

$\mathbf{Request_{Alice\to Bob}; (Accept_{Bob\to Alice} + Reject_{Bob\to Alice})}$

The following services can be retrieved:

$\mathbf{[\tau;\overline{Request}_{Bob};(Accept+Reject)]_{Alice} \mid}$
$\mathbf{[Request;(\tau;\overline{Accept}_{Alice}+\tau;\overline{Reject}_{Alice})]_{Bob}}$

Choreography implementations (with output persistent contracts)

- Projection modified: $[\![a_{r\to s}]\!]_t = \tau;\overline{a_s}$ if $t=r$

$\mathbf{Request_{Alice\to Bob}; (Accept_{Bob\to Alice} + Reject_{Bob\to Alice})}$

The following services can be retrieved:

$[\tau;\overline{Request_{Bob}};(Accept+Reject)]_{Alice} \mid [Request;(\tau;\overline{Accept_{Alice}}+\tau;\overline{Reject_{Alice}})]_{Bob}$

$\mathbf{[\tau;\overline{Request_{Bob}};(Accept+Reject+Retry)]_{Alice} \mid [Request;(\tau;\overline{Accept_{Alice}}+\tau;\overline{Reject_{Alice}})]_{Bob}}$

Choreography implementations (with output persistent contracts)

- Projection modified: $[\![a_{r\to s}]\!]_t = \tau;\overline{a}_s$ if $t=r$

$\mathbf{Request_{Alice\to Bob}; (Accept_{Bob\to Alice} + Reject_{Bob\to Alice})}$

The following services can be retrieved:

$[\tau;\overline{Request}_{Bob};(Accept+Reject)]_{Alice} \mid [Request;(\tau;\overline{Accept}_{Alice}+\tau;\overline{Reject}_{Alice})]_{Bob}$

$[\tau;\overline{Request}_{Bob};(Accept+Reject+Retry)]_{Alice} \mid [Request;(\tau;\overline{Accept}_{Alice}+\tau;\overline{Reject}_{Alice})]_{Bob}$

$\mathbf{[\tau;\overline{Request}_{Bob};(Accept+Reject+Retry)]_{Alice} \mid [Request;\tau;\overline{Accept}_{Alice}]_{Bob}}$

Compliance testing is the maximal preorder

- $C' \leq^{max} C$ iff for every context P $[C]|P$ compliance implies $[C']|P$ compliance
  - $\leq^{max}$ is a subcontract preoder
  - $\leq^{max}$ includes all subcontract preorders
- With respect to (fair) testing not only the test P has to succeed but also the tested C

Properties of the maximal subcontract preorder

- If we assume outputs $\overline{a_l}$ to be directed to a location l (e.g. role of a choreography) we can reduce the problem:

$$C' \leq^{max} C \text{ iff } C'\backslash N\text{-}I(C) \leq^{max} C$$

  i.e. to subcontract relation when inputs not among inputs of C $I(C)$ are restricted
  - because compliant tests of C cannot perform reachable outputs to C that it cannot receive

Thus…
(typical in session types)

- External choices on inputs can be extended:
  - $a+b \leq^{max} a$ (thanks to this property)
- Internal choices on outputs can be reduced:
  - $\tau.\overline{a_1} \leq^{max} \tau;\overline{a_1} + \tau;\overline{b_{1'}}$ (being more deterministic, as in testing)

Input and Output knowledge

- Contracts with undirected outputs à la CCS property does not hold, e.g. $\mathbf{a+b} \not\leq^{\max} \mathbf{a}$
  - Consider for instance (capturing)
    - the correct system $[\,\mathbf{a}\,] \mid [\tau;\overline{\mathbf{a}}.\mathbf{b}] \mid [\tau;\overline{\mathbf{b}}\,]$ and
    - the incorrect one $[\,\mathbf{a+b}\,] \mid [\tau;\overline{\mathbf{a}}.\mathbf{b}] \mid [\tau;\overline{\mathbf{b}}\,]$
- Problem can be solved by considering knowledge of I/O of other contracts $\leq_{I,O}$
  - exploiting knowledge we have $\mathbf{a+b} \leq_{\mathbf{N},\mathbf{N}-\{\mathbf{b}\}} \mathbf{a}$

Decidible Sound Characterization Based on a Must-like Testing

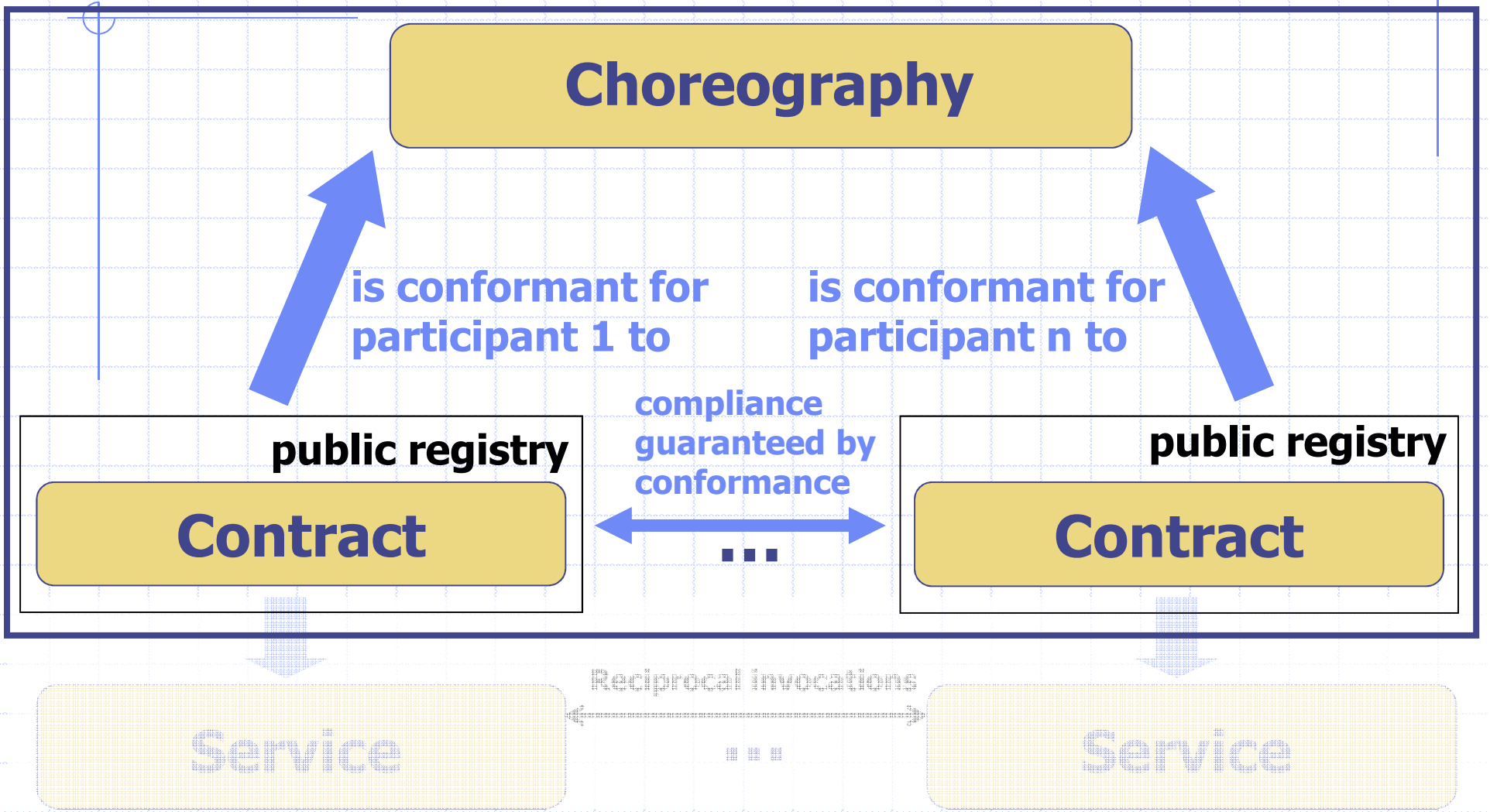
- Subcontract relation contains a universal quantification over possible contexts
- Sound characterization resorting to a must-testing theory (should-testing [RV05])
  - $C' \leq^{max} C$ is implied by

    $$C' \setminus N\text{-}I(C) \leq_{\text{should-testing}} C$$
  - i.e. $\leq^{max}$ coarser than testing preorder (and of simulation)

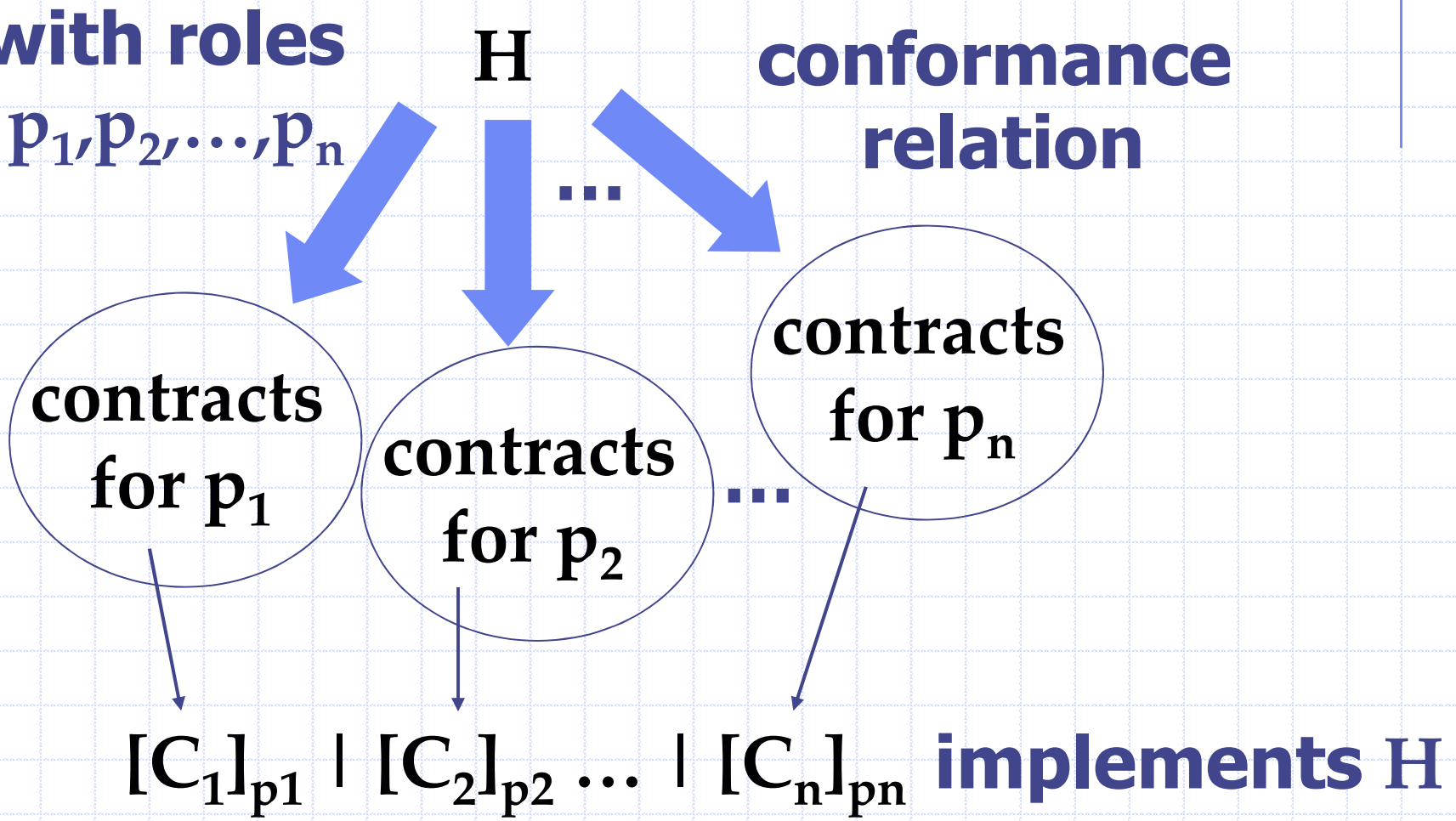
Uncontrollable contracts

- Trace equivalence not coarser than $\leq^{max}$ because contracts can include traces not leading to success
  - Those traces not observed by tests
  - Example:
    uncontrollable contracts (unsuccesful for any test) are all equivalent:
    - **a;b;0** equivalent to **b;c;0**

Choreography Conformance



Definition of Relation Induced from Independence Property



No Maximal Choreography Conformance Relation

- Consider the choreography

$$a_{r\to s} \mid b_{r\to s}$$

that can be implemented as:

$$[\ \tau;\overline{a}_s \mid \tau;\overline{b}_s\ ]_r \mid [\ \tau;a;b + \tau;b;a\ ]_s$$

$$[\tau;\overline{a}_s;\tau;\overline{b}_s + \tau;\overline{b}_s;\tau;\overline{a}_s\ ]_r \mid [a \mid b]_s$$

but not as:

$$[\tau;\overline{a}_s;\tau;\overline{b}_s + \tau;\overline{b}_s;\tau;\overline{a}_s\ ]_r \mid [\ \tau;a;b + \tau;b;a\ ]_s$$

  - Notice that we used output persistent contracts, so even asymmetry of I/O does not help

Summary of Results

- Refinement with knowledge about other initial contracts limited to I/O actions
  - "normal" compliance:
    - Uncostrained contracts: maximal relation does not exist
    - Contracts where outputs are internally chosen (output persistence): maximal relation exists and "I" knowledge is irrelevant
    - Output persistent contracts where outputs are directed to a location: maximal relation exists and "I/O" knowledge is irrelevant
  - strong compliance:
    - Uncostrained contracts (where output are directed to a location): maximal relation exists and "I/O" knowledge is irrelevant
  - queue-based (asynchronous) compliance:
    - Uncostrained contracts (where output are directed to a location): maximal relation exists and "I/O" knowledge is irrelevant

Summary of Results

- Direct conformance w.r.t. the whole choreography: maximal relation does not exist (all kinds of compl.)
- Sound characterizations of the relations obtained (apart from the queue based) by resorting to an encoding into (a fair version of) must testing [RV05]
  - With respect to testing: both system and test must succeed
  - Much coarser: all non-controllable systems are equivalent
- As a consequence:
  - Algorithm that guarantees compliance
  - Classification of the relations w.r.t. existing pre-orders: coarser than (fair) must testing (e.g., they allow external non-determinism on inputs to be added in refinements)

Plan of the Talk

- Choreographies and Orchestrations
- Contract-based service discovery
- A dynamic update mechanism
- Conclusion

Updatable processes/contracts
(with Marco Carbone)

- How to model updatable processes? Eg.
  - services which receive workflow from the environment in order to interact with it
  - internal "adaptable/mutable" subparts of cloud behaviour
- By extending choreographies and orchestrations/contracts with
  - updatable parts (named scopes) X[H] and
  - update actions/primitives X{H}

Buyer-Seller-Bank Example

- Consider the running system:

$$[X[C_{mcBuyer}] \mid C]_{buyer} \mid [C']_{seller} \mid [X[C_{mcBank}] \mid C'']_{bank}$$

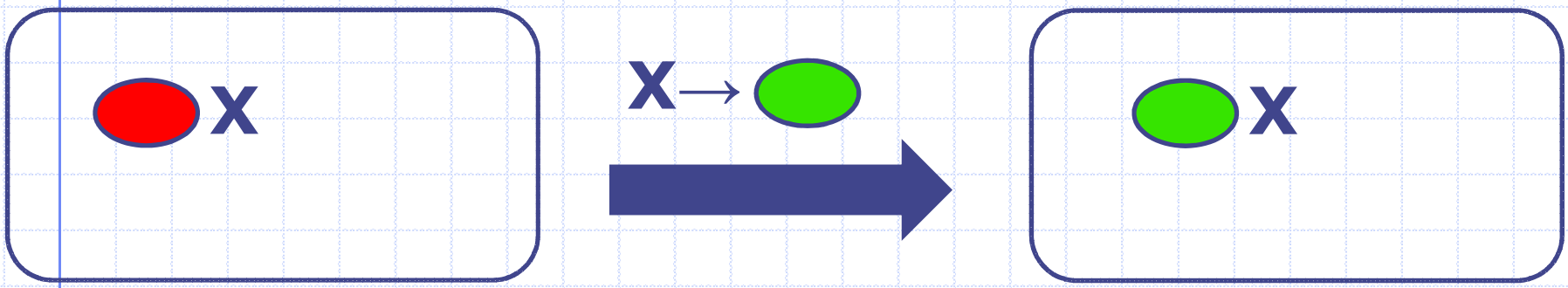
if the following update is performed:

$$X\{buyer:C_{visaBuyer},\ bank:C_{visaBank}\}$$

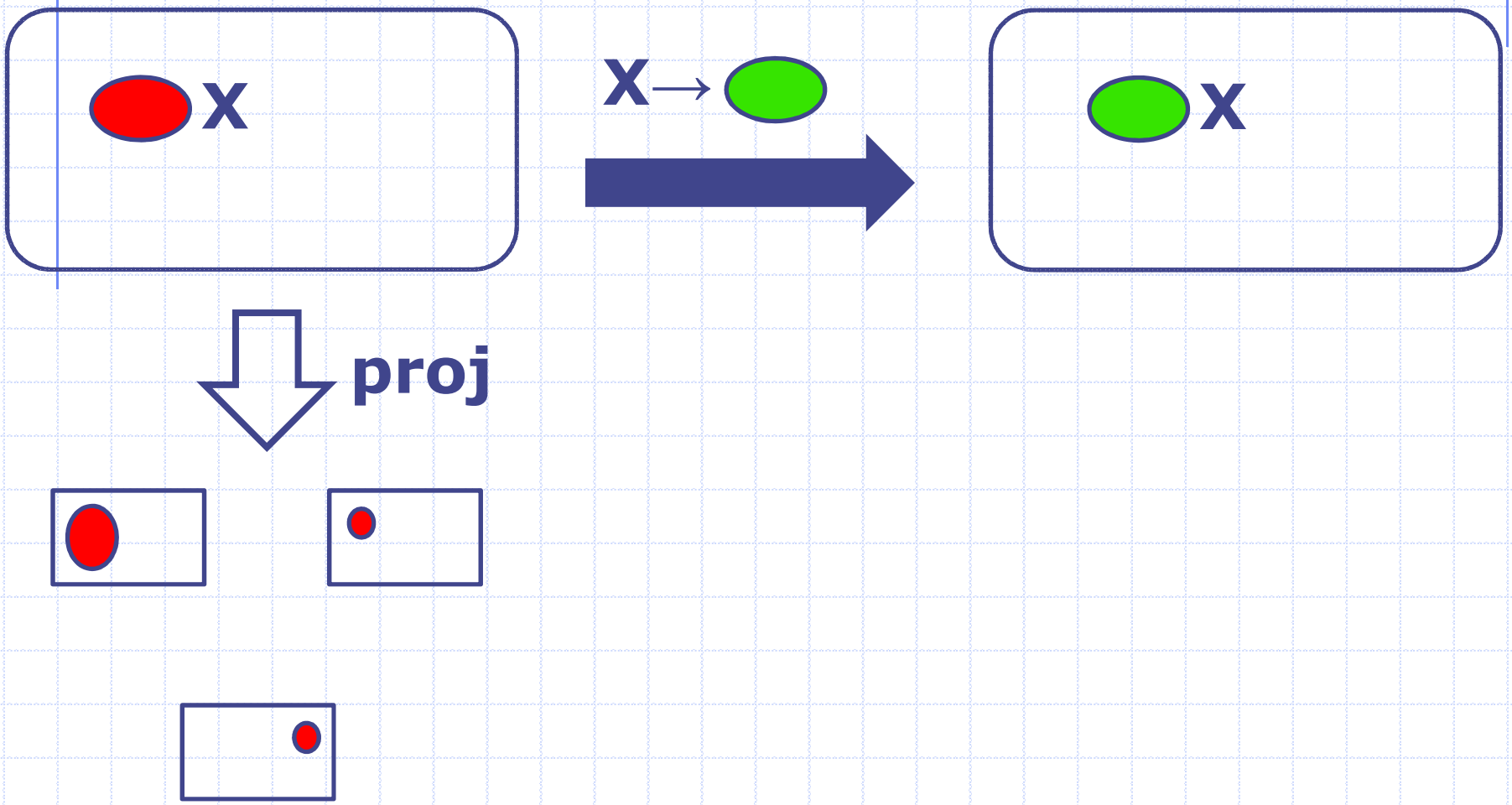
the system becomes:

$$[X[C_{visaBuyer}] \mid C]_{buyer} \mid [C']_{seller} \mid [X[C_{visaBank}] \mid C'']_{bank}$$

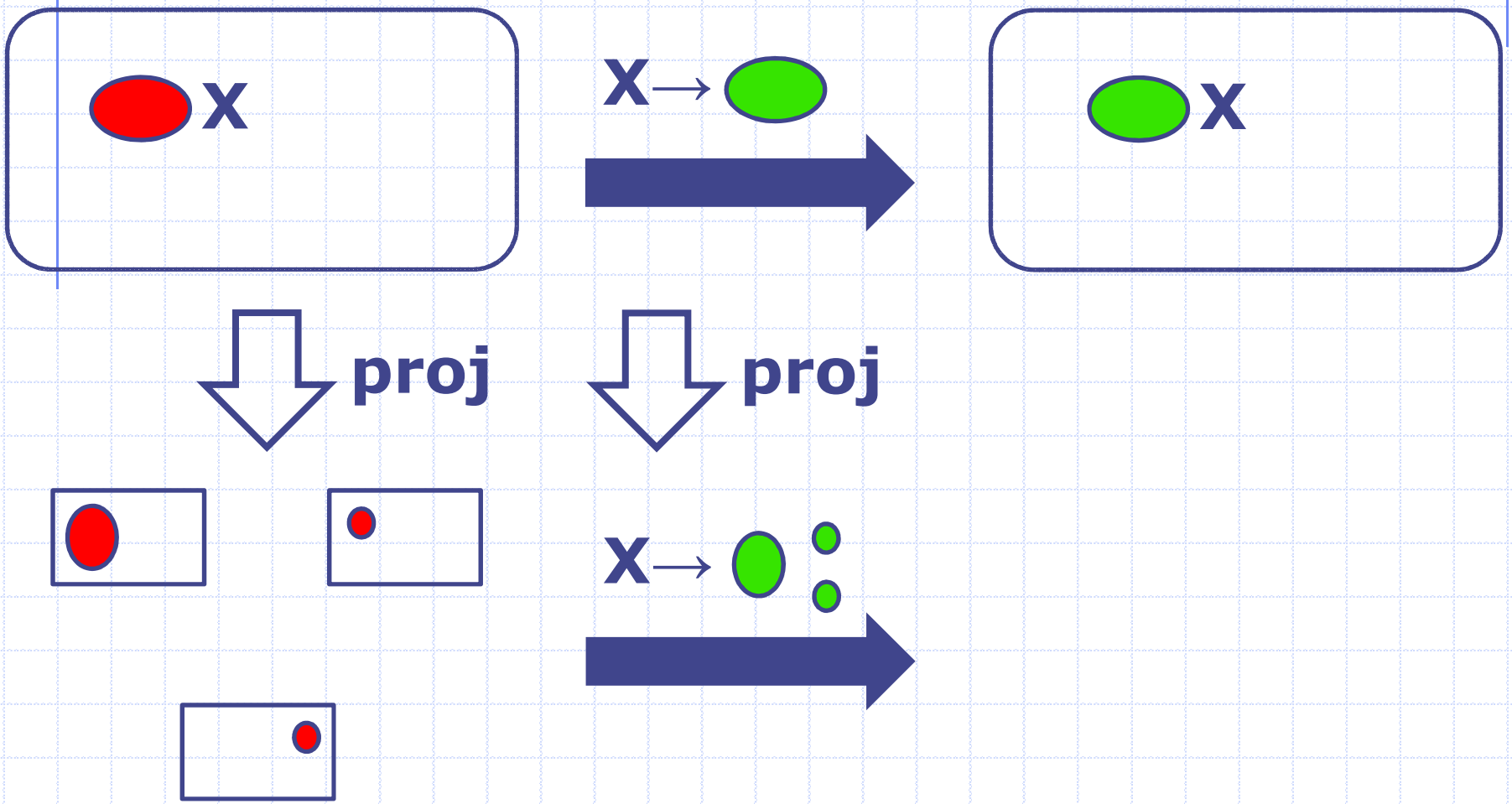
Updating a scope, graphically



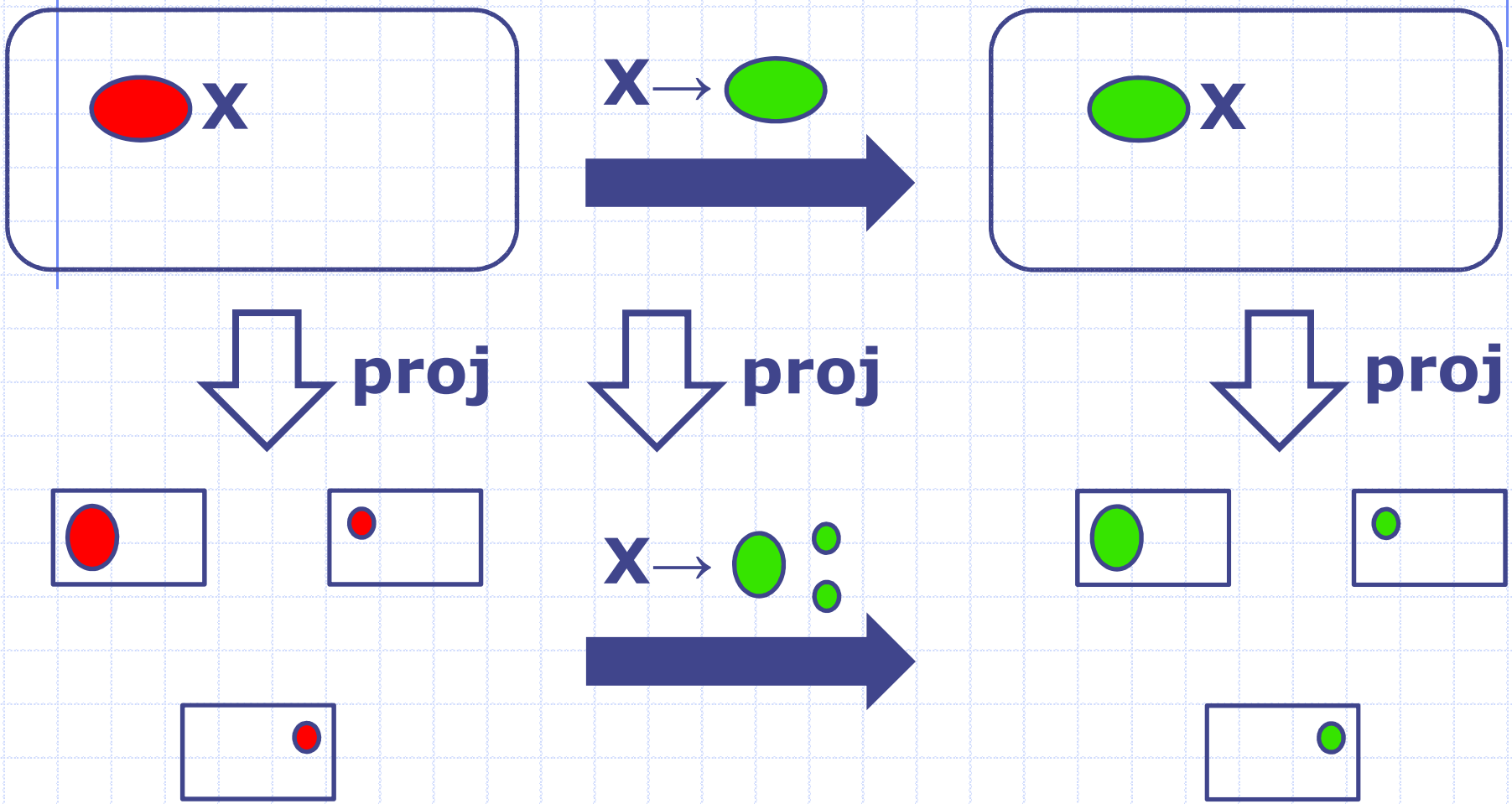
Updating a scope, graphically



Updating a scope, graphically



Updating a scope, graphically



Dynamic Choreographies

- Each scope X associated to a set of roles
  - given function **type(X)** = $\{r_1,\ldots,r_n\}$
- Global choreography language:

$$H ::= a_{r\to s} \mid 1 \mid 0 \mid H;H \mid H+H \mid H|H \mid H^* \mid X[H] \mid X_r\{H\}$$

with roles(H) included in type(X)

Extending choreography semantics with scope and updates

$$(\textsc{CommUpd})\ \frac{}{X_r\{H\} \xrightarrow{X_r\{H\}} \mathbf{1}} \qquad (\textsc{SeqUpd})\ \frac{H_1 \xrightarrow{X_r\{H\}} H_1'}{H_1;\ H_2 \xrightarrow{X_r\{H\}} H_1';(H_2[H/X])}$$

$$(\textsc{ParUpd})\ \frac{H_1 \xrightarrow{X_r\{H\}} H_1'}{H_1 \mid H_2 \xrightarrow{X_r\{H\}} H_1' \mid (H_2[H/X])} \qquad (\textsc{StarUpd})\ \frac{H_1 \xrightarrow{X_r\{H\}} H_1'}{H_1^* \xrightarrow{X_r\{H\}} H_1';\ (H_1[H/X])^*}$$

$$(\textsc{Scope})\ \ X[H] \xrightarrow{X^{\uparrow}} H$$

- where H[H'/X] turns scopes X[H''], for any H'', occurring inside H into X[H']

Dynamic Orchestrations

- A language for orchestrations:

$$C ::= a \mid \overline{a}_r \mid 1 \mid 0 \mid C;C \mid C+C \mid C|C \mid C^* \mid X[C] \mid X\{r_1:C_1, \ldots, r_n:C_n\}$$

$$P ::= [C]_r \mid P|P$$

- A distributed update can be performed if no scope X[C] of roles in type(X) has started

Standard "canonical" projection [Bravetti, Zavattaro SC 2007]

- Projection $[\![H]\!]_t$ of choreography H to participant t

$$[\![a_{r \to s}]\!]_t = \begin{cases} \overline{a_s} & \text{if } t=r \\ a & \text{if } t=s \\ 1 & \text{otherwise} \end{cases}$$

$$[\![H;H']\!]_t=[\![H]\!]_t ; [\![H']\!]_t \qquad [\![H \mid H']\!]_t=[\![H]\!]_t \mid [\![H']\!]_t$$

$$[\![H+H']\!]_t=[\![H]\!]_t + [\![H']\!]_t \qquad [\![H^*]\!]_t=[\![H]\!]_t^*$$

Extended….

- Projection $[[\ H\ ]]_t$ of choreography **H** to participant **t**

$$[[\ X[H]\ ]]_t = \begin{cases} X[\ [[H]]_t\ ] & \text{if } t \text{ in } type(X) \\ 1 & \text{otherwise} \end{cases}$$

$$[[\ X_r\{H\}\ ]]_t = \begin{cases} X\{t_1:[[H]]_{t1}, \ldots, t_n:[[H]]_{tn}\} & \text{if } t = r \\ 1 & \text{otherwise} \end{cases}$$

where $type(X) = \{t_1,\ldots,t_n\}$

Extension of Connectedness and external updates

- Constraint: there are not (and cannot be produced) scopes in parallel with the same name X
  - so not to confuse starts of different scopes
- Open transitions: update choreographies H in a scope X from "outside" provided:
  - H is connected
  - The update does not violate the constraint above

Main Theorem

- Projection of a connected choreography H produces an implementation of H
  - Traces considered by implementation definition also include open transitions

Typing a concrete langauge with sessions (session types)

- Channels "**a**" statically associated to choreographies H (global type)

$$
\begin{array}{lr}
P ::= \texttt{start}\ a_{\mathbf{r}}(s).P & \text{session start} \\
\quad \mid s.\texttt{send}[op]\langle e\rangle & \text{send} \\
\quad \mid s.\texttt{pick}\{\omega_i\ op_i(x_i) : P_i\}_{i\in I} & \text{pick} \\
\quad \mid \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q & \text{conditional} \\
\quad \mid \texttt{while}\ e\ \texttt{do}\ P & \text{while} \\
\quad \mid P; P & \text{sequence} \\
\quad \mid P \mid P & \text{parallel composition} \\
\quad \mid \texttt{skip} & \text{skip} \\
\quad \mid X^{\tilde{s}}[P] & \text{scoped code} \\
\quad \mid X_{\tilde{s}}\{l_i[\tilde{z}_i] : P_i\}_{i\in I} & \text{update}
\end{array}
$$

Networks and Expressions

$\omega ::= \texttt{on} \mid \texttt{onrep}$ — pick events

$e ::= v \mid e \texttt{ and } e' \mid \texttt{not } e \quad \ldots$ — expressions

$v ::= s \mid \texttt{true} \mid \texttt{false}$ — values

$N ::= l :: P$ — location

$\mid N \parallel N$ — network parallel

Session Typing is not trivial

$$\Gamma \vdash P \triangleright \Delta$$

- Γ typing environment
  - each channel is typed with a choreography
- Δ maps:
  - each started session into an orchestration

Typing scopes and updates

- Inside a scope X[] we may communicate using several started sessions
- We must update all session (types) the process in the scope is engaged in
- Updates are going to update code inside multiple sessions
  - must be allowed individually by their types

An idea of the typing rules

$$\frac{\Gamma, a : H \vdash P \triangleright \Delta \cdot s@\mathtt{r} : [\![H]\!]_r}{\Gamma, a : H \vdash \mathtt{start}\ a_{\mathtt{r}}(s).P \triangleright \Delta} \ [\top/\mathsf{Start}]$$

$$\frac{\Gamma \vdash e : S}{\Gamma \vdash s.\mathtt{send}[op]\langle e \rangle \triangleright \Delta \cdot s@\mathtt{r} : \overline{op}\langle S \rangle} \ [\top/\mathsf{Send}]$$

Example with two channels

$$Roles(a) = \{\texttt{buyer}, \texttt{seller}, \texttt{bank}\}$$

$$Roles(b) = \{\texttt{cashInt}, \texttt{cash}\}$$

$$Network: \quad l_1 :: P_1 \parallel l_2 :: P_2 \parallel l_3 :: P_3 \parallel l_4 :: P_4 \parallel l_5 :: P_5$$

$$P_1 = \ldots \quad \texttt{start}\ a_{\texttt{buyer}}(s).\ (X^s[P_{mcBuyer}] \mid P) \quad \ldots$$

$$P_2 = \ldots \quad \texttt{start}\ a_{\texttt{seller}}(s).P' \quad \ldots$$

$$P_3 = \ldots \quad \texttt{start}\ a_{\texttt{bank}}(s).\ \texttt{start}\ b_{\texttt{cashInt}}(s').\ (X^{s,s'}[P_{mcBank\&CashInt}] \mid P'') \quad \ldots$$

$$P_4 = \ldots \quad \texttt{start}\ b_{\texttt{cash}}(s).\ X^s[P_{mcCash}] \quad \ldots$$

$$P_5 = \ldots \quad X\{l_1[s] : P_{visaBuyer}, \quad l_3[s,s'] : P_{visaBank\&CashInt}, \quad l_4[s] : P_{visaCash}\} \quad \ldots$$

Typing the example

Global Types: H_a, H_b

Local Types:

$$\begin{aligned}
[\![H_a]\!]_{buyer} &= X[C_{mcBuyer}] \mid C \\
[\![H_a]\!]_{seller} &= C' \\
[\![H_a]\!]_{bank} &= X[C_{mcBank}] \mid C''
\end{aligned}$$

$$\begin{aligned}
[\![H_b]\!]_{cashInt} &= X[C_{mcCashInt}] \\
[\![H_b]\!]_{cash} &= X[C_{mcCash}]
\end{aligned}$$

Typing:

$$\forall i \in \{1,2,3,4\}. \qquad a : H_a, b : H_b \vdash P_i \triangleright \emptyset$$

For instance: typing of P_3 reduces to

$$a : H_a, b : H_b \vdash X^{s,s'}[P_{mcBank\&CashInt}] \mid P'' \triangleright s@\mathsf{bank} : [\![H_a]\!]_{bank} \cdot s'@\mathsf{cashInt} : [\![H_b]\!]_{cashInt}$$

Plan of the Talk

- Choreographies and Orchestrations
- Contract-based service discovery
- A dynamic update mechanism
- Conclusion

Conclusion

- We are working on:
  - Use of the above in the context of session types for typing a concrete language: typing rules and properties (e.g. subject reduction)

Conclusion

- Open problems:
  - Complete characterization of compliance testing
    - work in this direction has been done in [Bernardi, Hennessy Concur 2013] where however uncontrollable processes are not characterized and fairness is not considered
  - Refinement theory for dynamic (with updates) choreographies/contracts

References

- M. Bravetti and G. Zavattaro. Contract based Multi-party Service Composition. In FSEN'07. (full version in Fundamenta Informaticae)
- M. Bravetti and G. Zavattaro. Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In SC '07.
- M. Bravetti and G. Zavattaro. A Theory for Strong Service Compliance. In Coordination'07. (full version in MSCS)
- M. Bravetti and G. Zavattaro. Contract Compliance and Choreography Conformance in the presence of Message Queues.In WS-FM '08
- M. Bravetti and G. Zavattaro. On the Expressive Power of Process Interruption and Compensation. In WS-FM '08
- M. Bravetti, I. Lanese, G. Zavattaro. Contract-Driven Implementation of Choreographies.In TGC '08
- M. Bravetti, G. Zavattaro. Contract-Based Discovery and Composition of Web Services. In Formal Methods for Web Services, Advanced Lectures '09, LNCS 5569
- M. Bravetti, C. Di Giusto, J. A. Pérez, and G. Zavattaro. A Calculus for Component Evolvability (Extended Abstract). In FACS'10
- M. Bravetti, C. Di Giusto, J. A. Pérez, and G. Zavattaro. Adaptable Processes (Extended Abstract). In FORTE/FMOODS'11
- M. Boreale, M. Bravetti. Advanced Mechanisms for Service Composition, Query and Discovery in Rigorous Software Eng. for Service-Oriented Systems '11, LNCS 6582
- M. Bravetti, M. Carbone, T. Hildebrandt, I. Lanese, J. Mauro, J. A. Pérez, G. Zavattaro: Towards Global and Local Types for Adaptation. In BEAT2 '13, LNCS 8368