Choreography Projection and Contract Refinement

Mario Bravetti
http://cs.unibo.it/~bravetti

Department of Computer Science
University of Bologna

INRIA research team FOCUS

Joint work with:
Ivan Lanese, Gianluigi Zavattaro

Plan of the Talk

- Global and Local Choreography
- 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

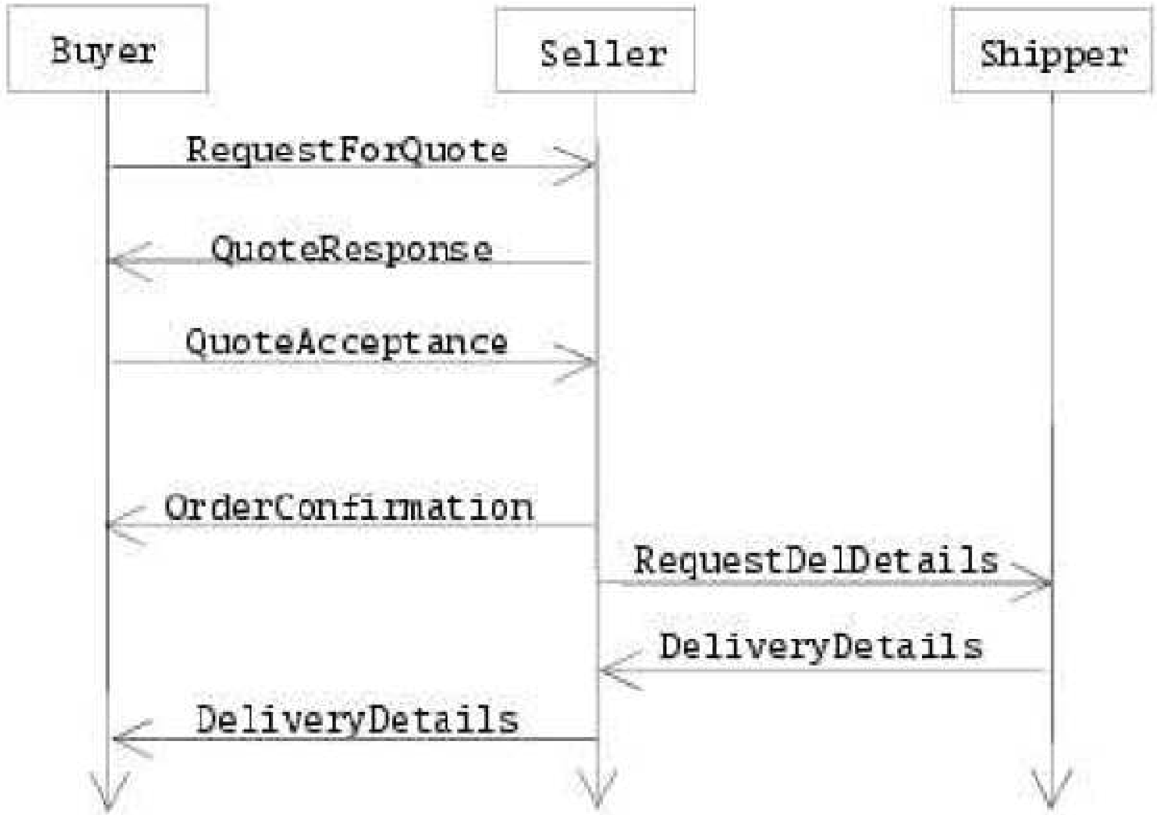
Choreography (e.g. WS-CDL)

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

Orchestration (e.g. WS-BPEL)

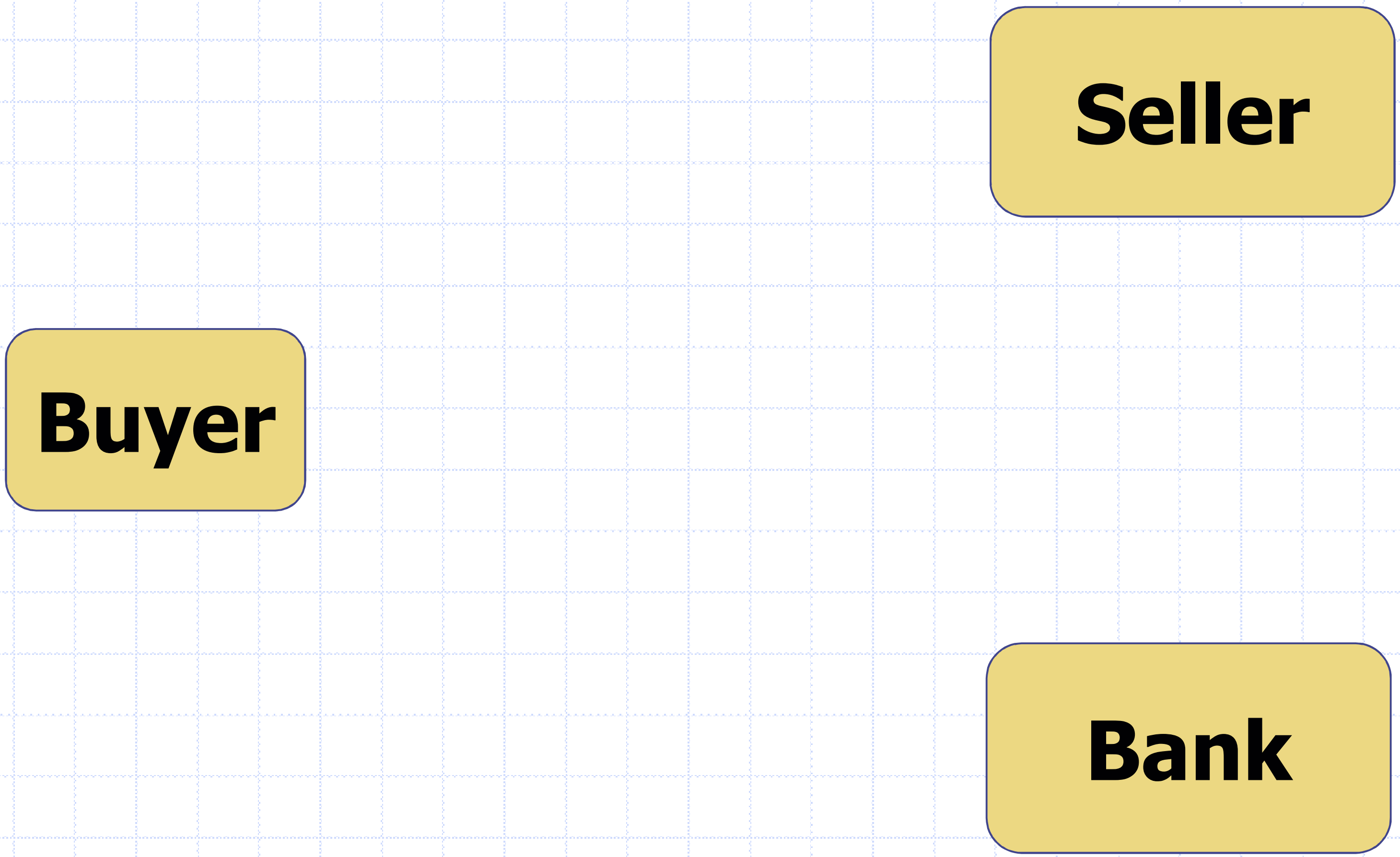
One Party detailed view of the system that orchestrates a part of it by sending (to other parties) & receiving messages

Similar to UML Sequence Diagrams



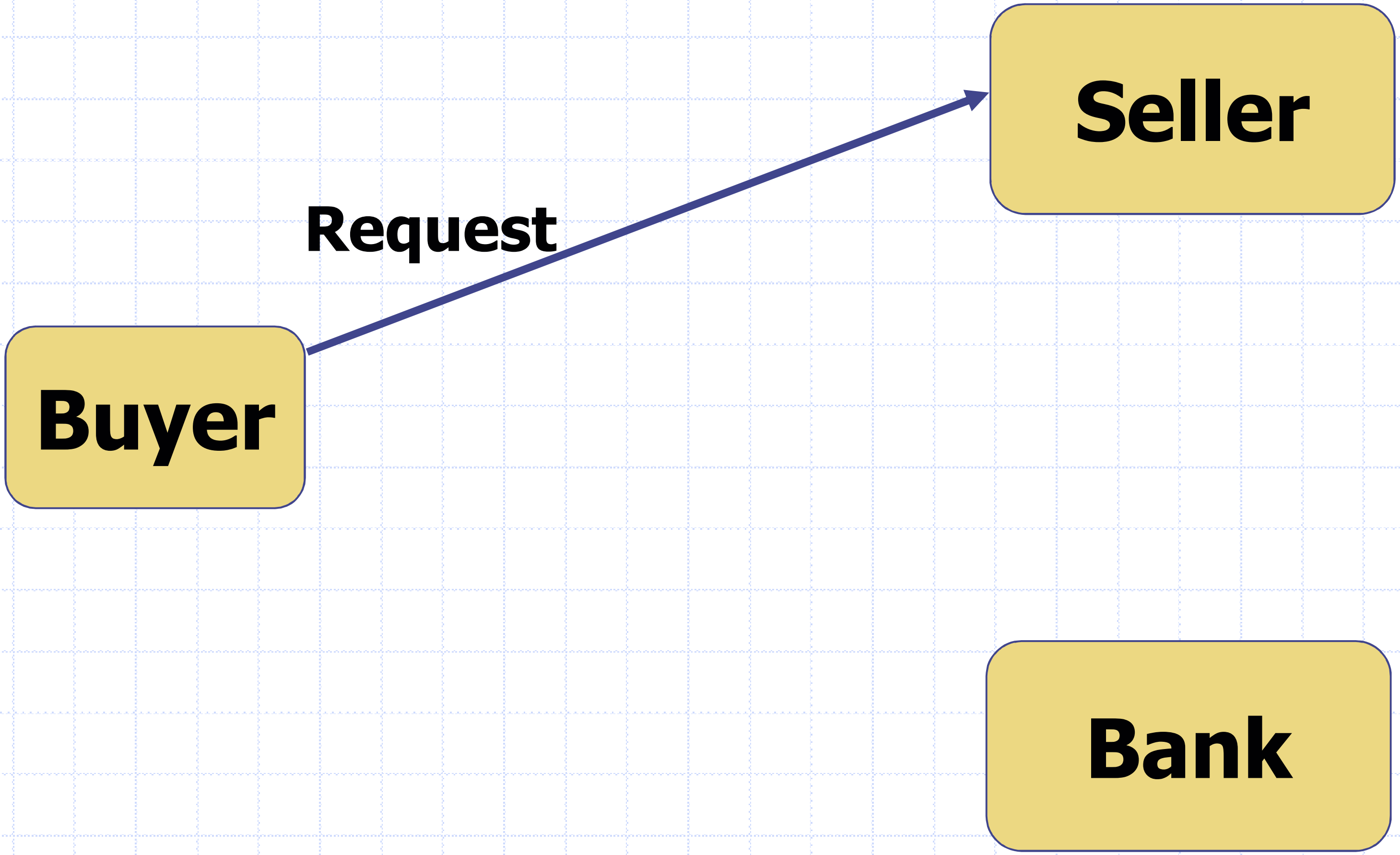
WS-CDL

- Global view of service interactions



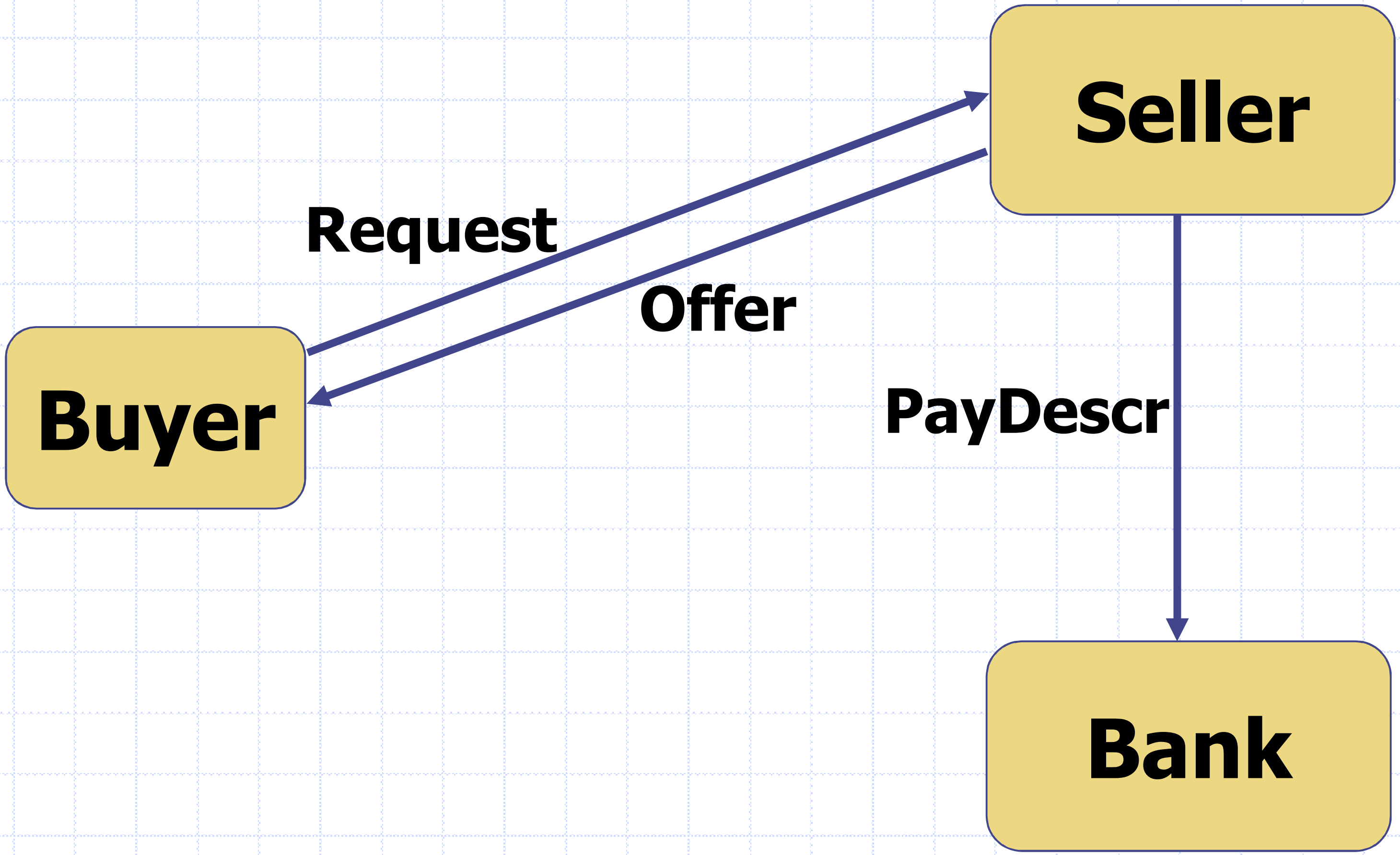
WS-CDL

- Global view of service interactions



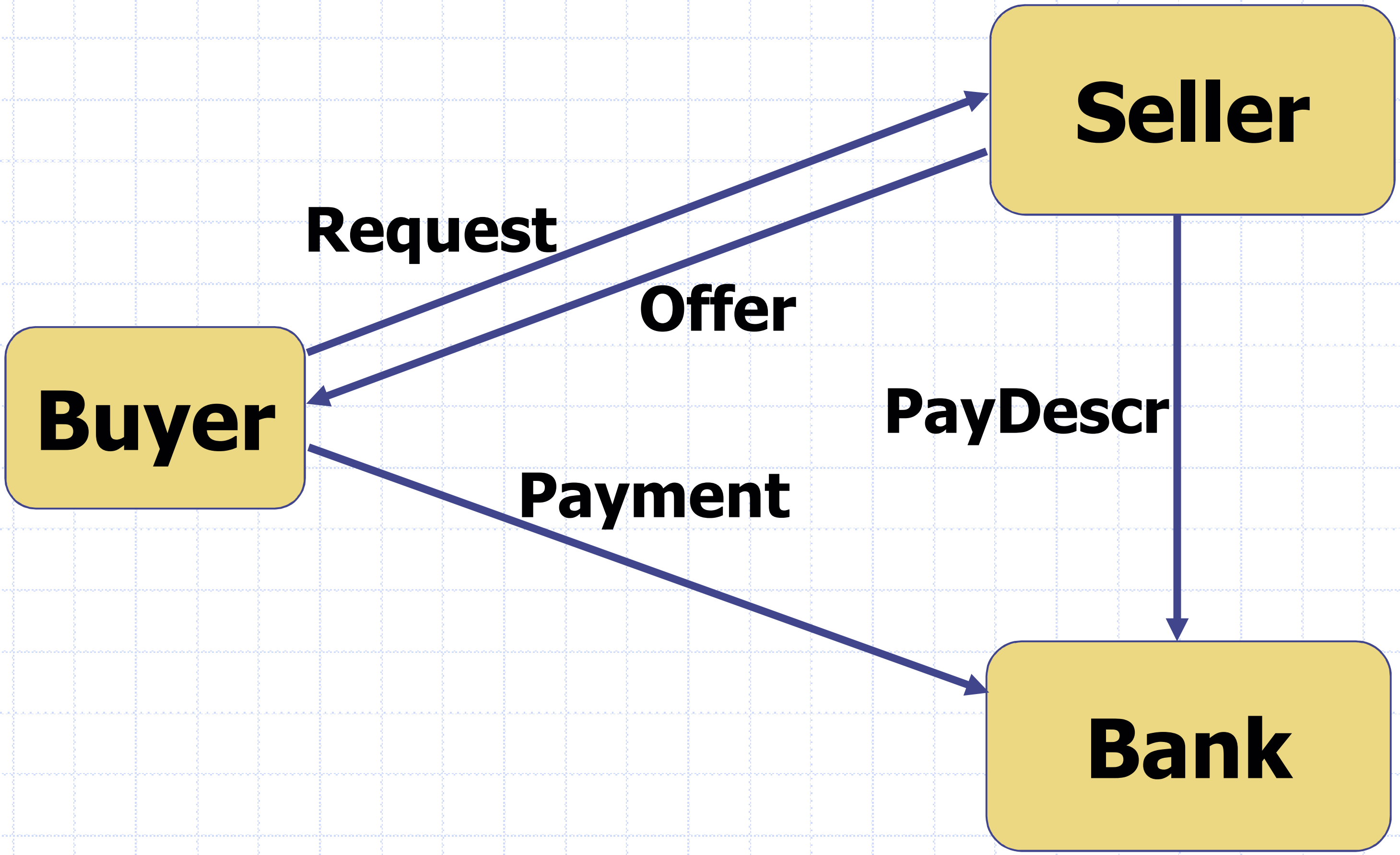
WS-CDL

- Global view of service interactions



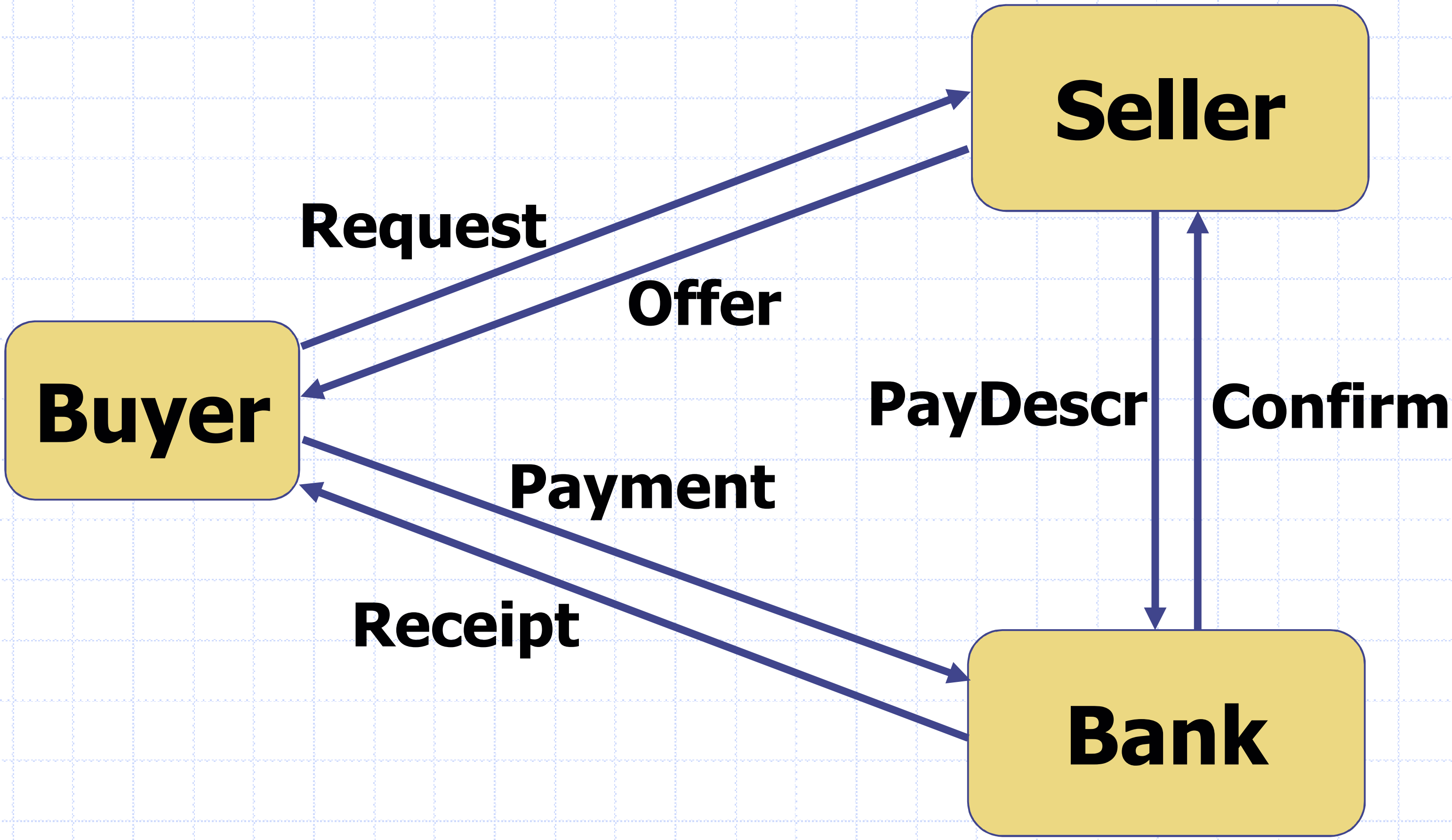
WS-CDL

- Global view of service interactions



WS-CDL

- Global view of service interactions



WS-CDL

$\text{Request}_{\text{Buyer}\rightarrow\text{Seller}}$;

($\text{Offer}_{\text{Seller}\rightarrow\text{Buyer}}$ |

$\text{PayDescr}_{\text{Seller}\rightarrow\text{Bank}}$) ;

$\text{Payment}_{\text{Buyer}\rightarrow\text{Bank}}$;

($\text{Confirm}_{\text{Bank}\rightarrow\text{Seller}}$ |

$\text{Receipt}_{\text{Bank}\rightarrow\text{Buyer}}$)

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)

Well Formed WS-CDL specifications

- Can we always project a WS-CDL specification in an equivalent one?
- Which kind of equivalences are preserved?

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:

$$
\begin{aligned}
H \quad ::= \quad & a_{r\to s} \mid 1 \mid 0 \mid \\
& H;H \mid H+H \mid H|H \mid H^*
\end{aligned}
$$

Sequence

Choice

Parallel

Repetition

A Formal Model for orchestrations

- A language for orchestrations:

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

$$S \;::=\; [P]_r \mid S|S$$

A Formal Model for orchestrations

- A language for orchestrations:

$$P \ ::= \quad a \mid \overline{a}_r \mid 1 \mid 0 \mid$$
$$\qquad P;P \mid P+P \mid P|P \mid P^*$$
$$S \ ::= \quad [P]_r \mid S|S$$

receive on a

invoke a at r

Successful termination

Unsuccessful termination

A Formal Model for orchestrations

- A language for orchestrations:

$$P \quad ::= \quad a \mid \overline{a}_r \mid 1 \mid 0 \mid$$
$$P;P \mid P+P \mid P|P \mid P^*$$
$$S \quad ::= \quad [P]_r \mid S|S$$

Sequence

Choice

Parallel

Repetition

A Formal Model for orchestrations

- A language for orchestrations:

$$P \quad ::= \quad a \mid \overline{a}_r \mid 1 \mid 0 \mid$$
$$P;P \mid P+P \mid P|P \mid P^*$$
$$S \quad ::= \quad [P]_r \mid S|S$$

Behaviour of participant r

Parallel composition of participants

The "canonical" projection

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

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

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

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

Example

- Consider the global choreography:

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

- Projection:

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

- Are the two choreographies equivalent?
  - NO
  - But, if r=t.... YES

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

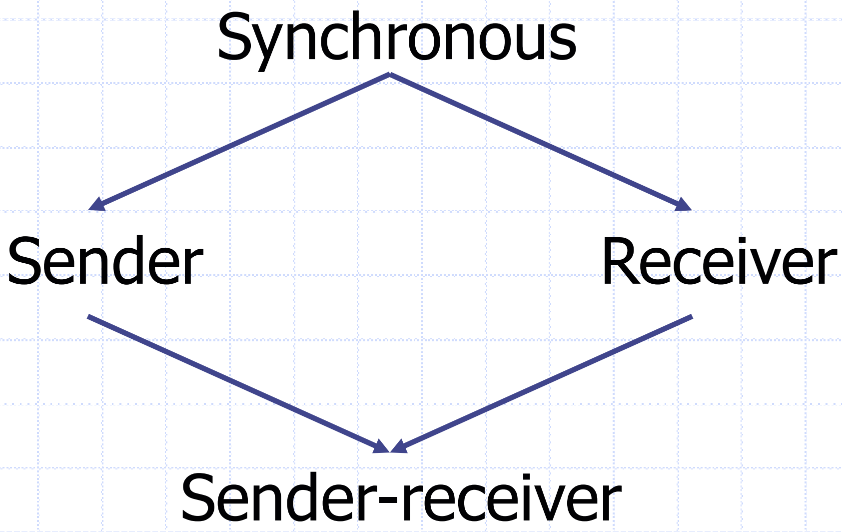
Asynchronous communication

- Reconsider the example assuming asynchronous communication

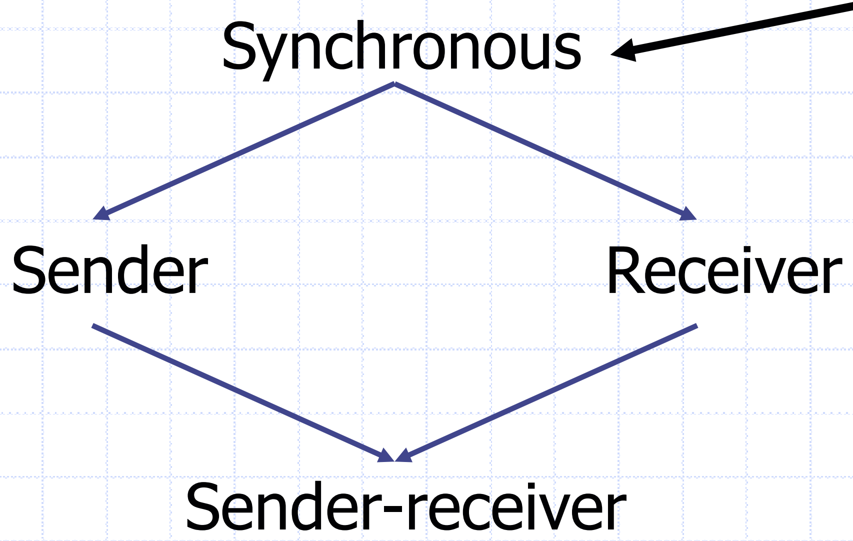
  $[\,\overline{a}_s;\overline{b}_u\,]_r \mid [\,a\,]_s \mid [\,b\,]_u$

- Communication on **a** starts before communication on **b** but could finish after
- What we should observe?
  - Send, Receive, both, …?

A lattice of possible observation criteria

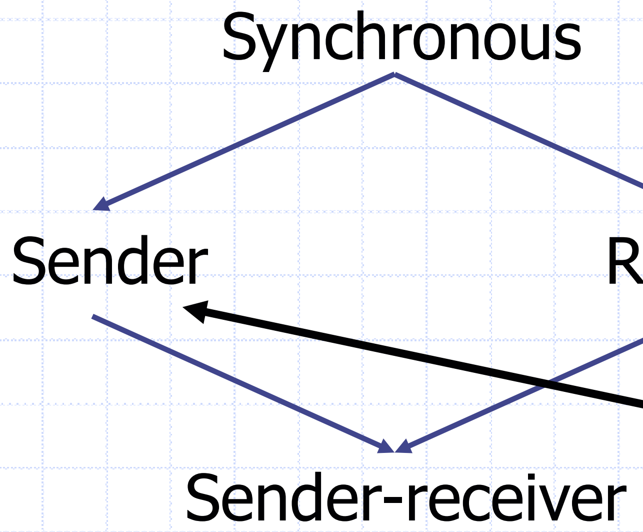


A lattice of possible observation criteria



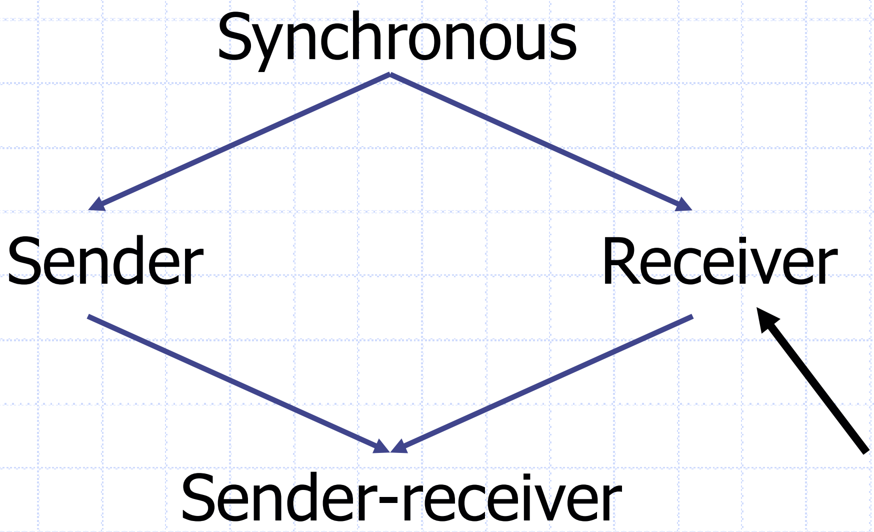
Assuming synchronous communication: observe either send or receive

A lattice of possible observation criteria



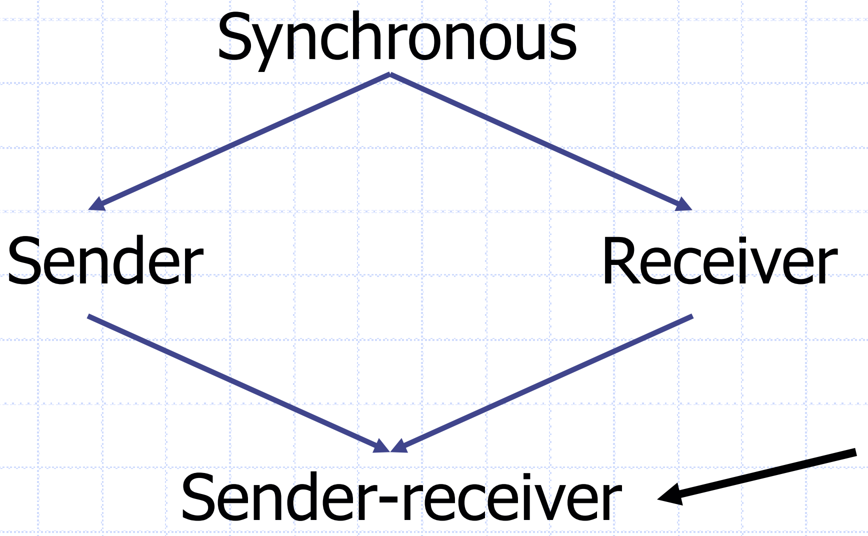
Assuming asynchronous communication: observe send

A lattice of possible observation criteria



Assuming asynchronous communication: observe receive

A lattice of possible observation criteria



Assuming asynchronous communication: observe send and observe receive

What about the previous example?

- Reconsider the example

$$a_{r\rightarrow s} ; b_{r\rightarrow u}$$

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

- OK: for synchronous and sender
- NO: for receiver, sender-receiver

Main results

- For each observation criterion:
  - Sufficient conditions (connectedness, unique point of choice, and causality safe) that guarantee that a global choreography is equivalent to the projected one

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

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

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

Which equivalence between global and local choreographies?

- Synchronous equivalence: global transitions are matched by synchronous local transitions
- Sender equivalence: global transitions are matched by local sends, local receives are abstracted away
  - weak w.r.t. local receive transitions
- Receiver equivalence: global transitions are matched by local receives, global sends are abstracted away
  - weak w.r.t. local send transitions
- Sender-Receiver equivalence: both conditions above

Example: Receiver equivalence

- Global choreography:

  $$\mathbf{a}_{r \to s} ; \mathbf{b}_{t \to s}$$

- Local choreography:

  $$[\ \overline{\mathbf{a}}_s\ ]_r \mid [\ \mathbf{a};\mathbf{b}\ ]_s \mid [\ \overline{\mathbf{b}}_s\ ]_t$$

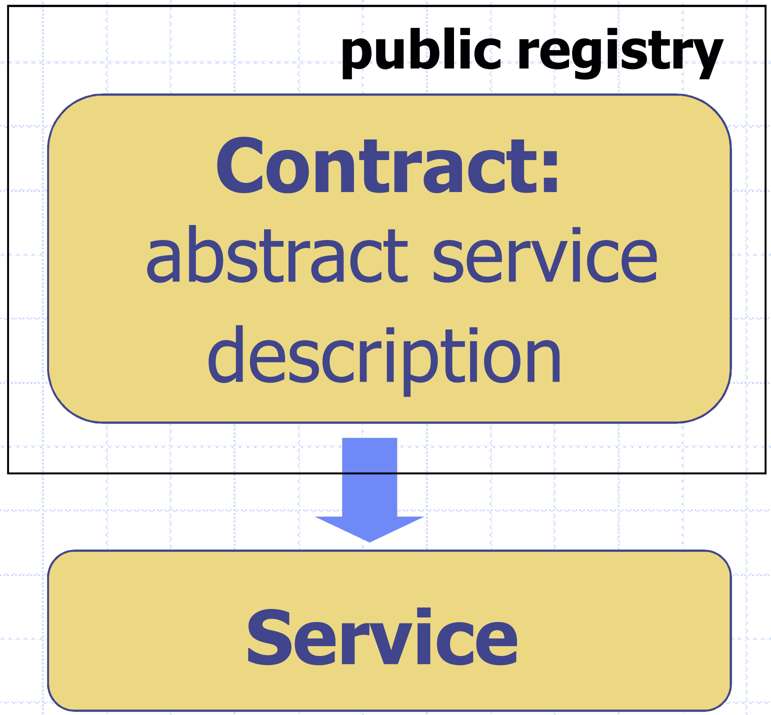
- The two systems are receiver equivalent

Plan of the Talk

- Global and Local Choreography
- Contract-based service discovery
- A dynamic update mechanism
- Conclusion

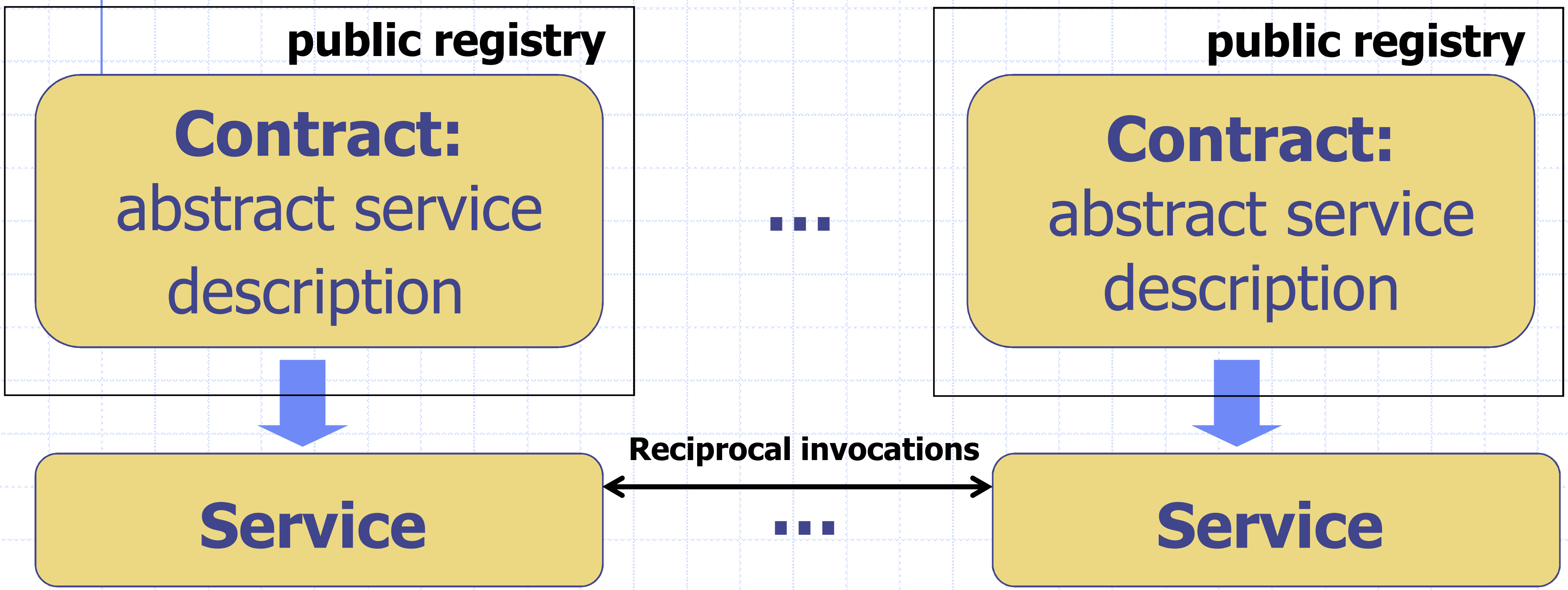
Contracts

- 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 **P**:

  $$\mathbf{P} \overset{\tau}{\dashrightarrow}{}^{*} \mathbf{P'}$$

  implies that there exist **P″** and **P‴** s.t.

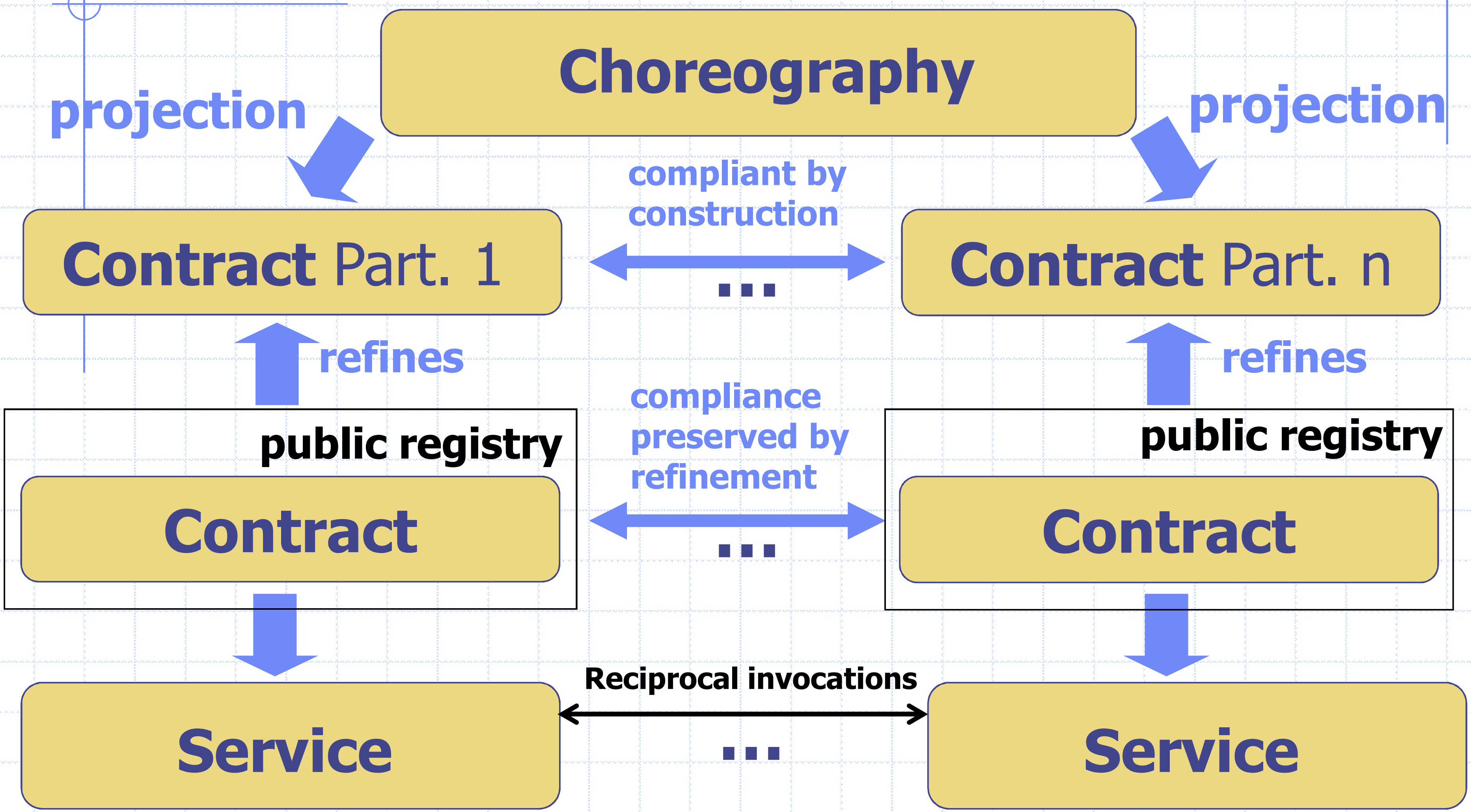
  $$\mathbf{P'} \overset{\tau}{\dashrightarrow}{}^{*} \mathbf{P''} \overset{\surd}{\dashrightarrow} \mathbf{P'''}$$

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

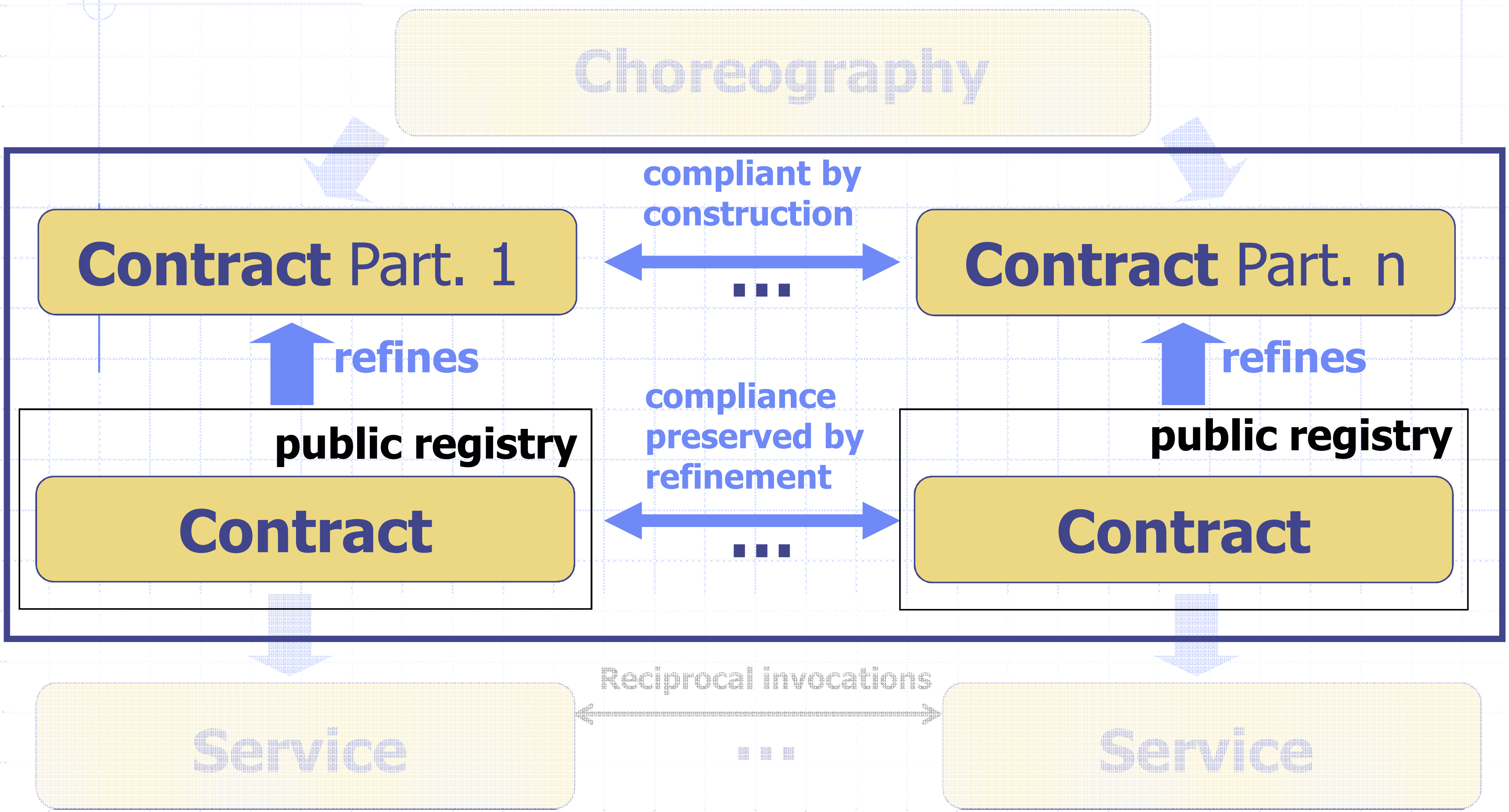
Example: compliant services

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

Compliance-Preserving Contract Refinement !

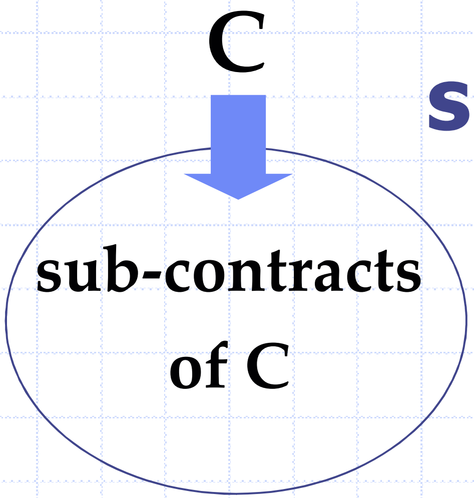


Contract Refinement Relation



Formally: Subcontract Preorder

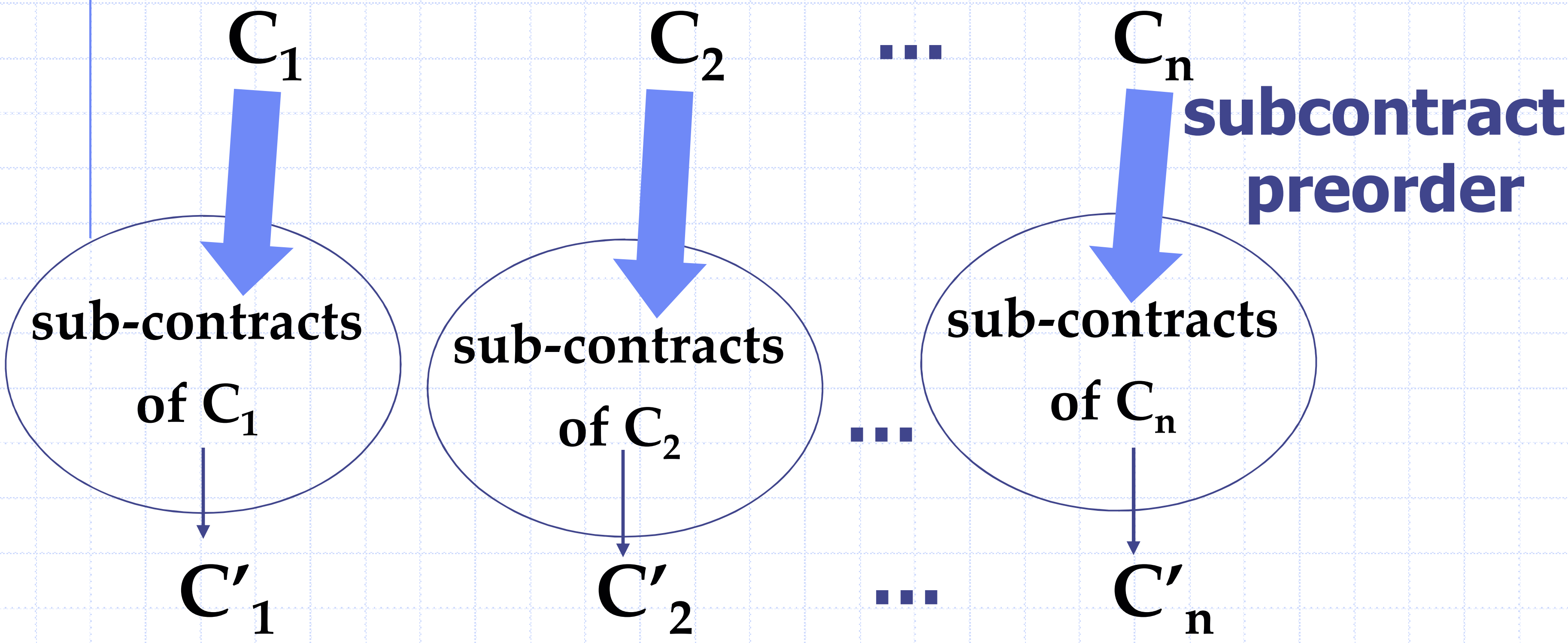
- Preorder ≤ between contracts C:
  - C′ ≤ C means C′ is a subcontract of C



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:
  - Limiting the considered services (output persistence)
  - Strengthening the notion of compliance (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 action on **a** and $\mathbf{P} \xrightarrow{\alpha} \mathbf{P'}$ with α different from output on **a**, then also **P'** has an output on **a**
- This holds, for instance, in WS-BPEL
  - Outputs cannot resolve the pick operator for external choices (the decision to execute outputs is taken internally)

Example

- Given the choreography:

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

The following services can be retrieved:

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

Example

- Given the choreography:

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

The following services can be retrieved:

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

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

Example

- Given the choreography:

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

The following services can be retrieved:

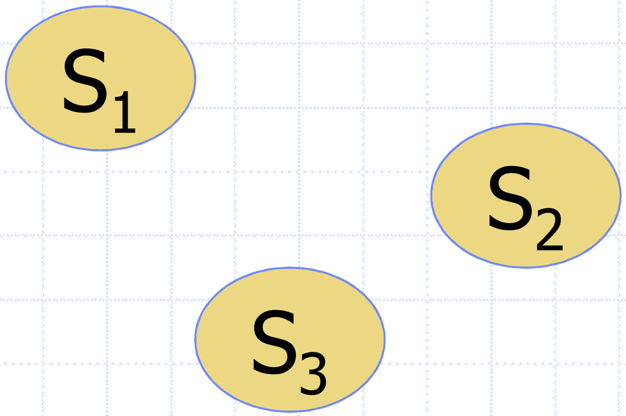
$$[\tau;\overline{\text{Request}}_{Bob};(\text{Accept}+\text{Reject})]_{Alice} \mid$$
$$[\text{Request};(\tau;\overline{\text{Accept}}_{Alice}+\tau;\overline{\text{Reject}}_{Alice})]_{Bob}$$

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

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

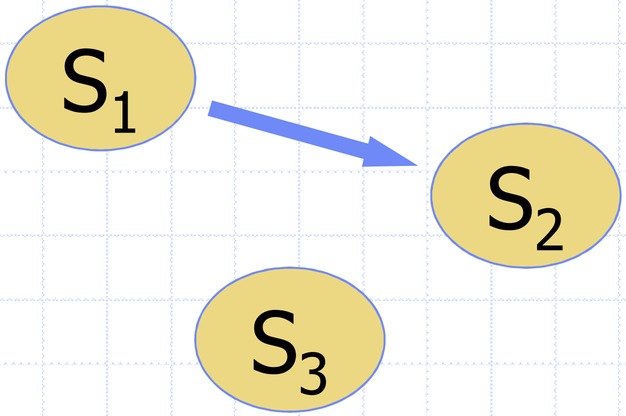
"Standard" Contract Compliance

- Example:
  - **S_1:** invoke(a);invoke(b)
  - **S_2:** receive(a);invoke(c)
  - **S_3:** receive(c);receive(b)



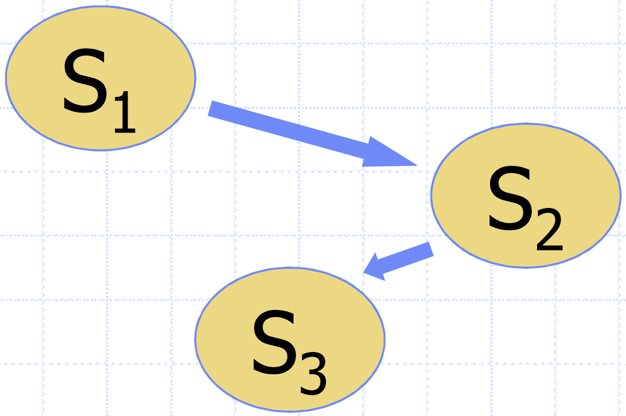
"Standard" Contract Compliance

- Example:
  - S_1: **invoke(a);**invoke(b)
  - S_2: **receive(a);**invoke(c)
  - S_3: receive(c);receive(b)



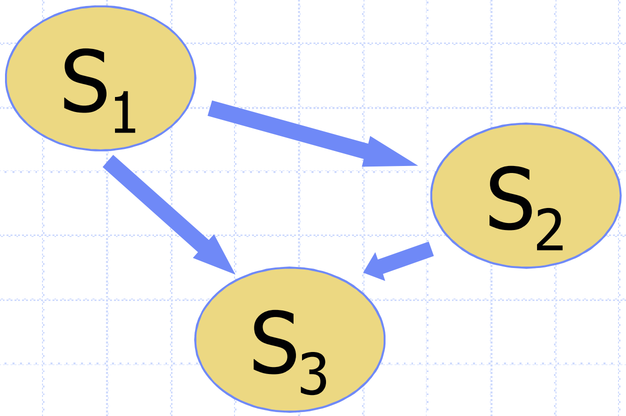
"Standard" Contract Compliance

- Example:
  - S_1: invoke(a);invoke(b)
  - S_2: receive(a);invoke(c)
  - S_3: receive(c);receive(b)



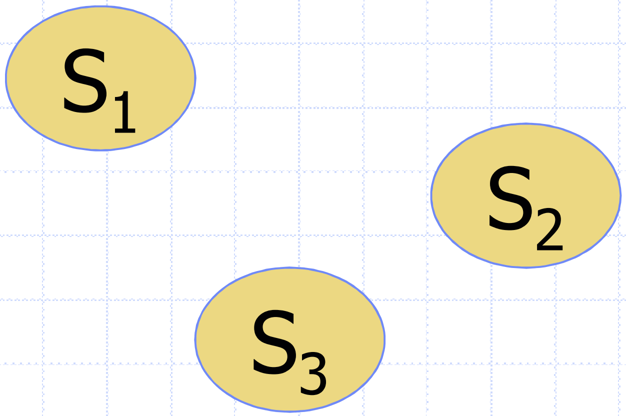
“Standard” Contract Compliance

- Example:
  - **S_1: invoke(a);invoke(b)**
  - **S_2: receive(a);invoke(c)**
  - **S_3: receive(c);receive(b)**



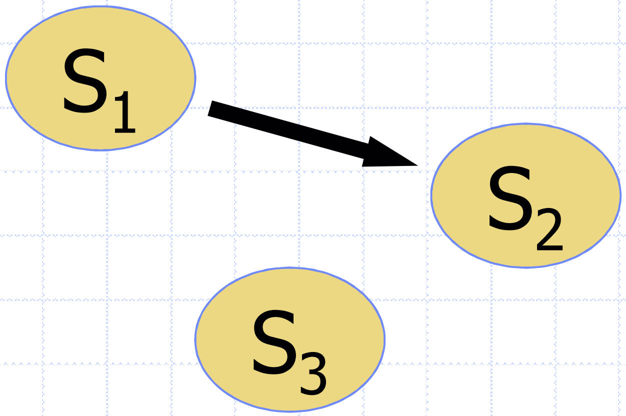
Alternatives to Standard Compliance: Strong Compliance

- Let us give a more careful look:
  - S_1: invoke(a);invoke(b)
  - S_2: receive(a);invoke(c)
  - S_3: receive(c);receive(b)



Alternatives to Standard Compliance: Strong Compliance

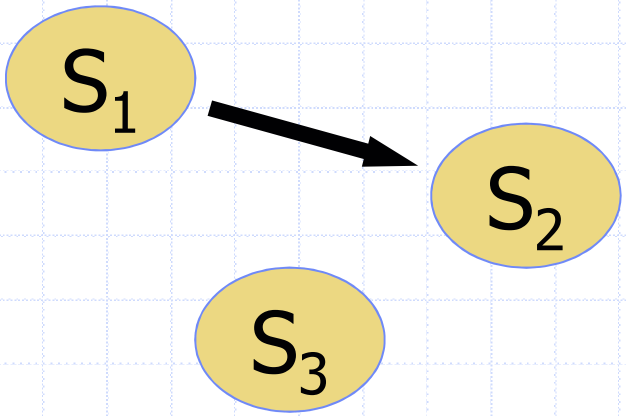
- Let us give a more careful look:
  - S_1: invoke(a);invoke(b)
  - S_2: receive(a);invoke(c)
  - S_3: receive(c);receive(b)



Alternatives to Standard Compliance: Strong Compliance

- Let us give a more careful look:
  - S_1: invoke(a);invoke(b)
  - S_2: receive(a);invoke(c)
  - S_3: receive(c);receive(b)

- These services are not **strongly** compliant !!



- **Strong compliance** requires that the receptors should be always ready

Example: strong compliant services

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

Example: strong compliant services

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

"Strong" refinement

- It allows also refinement on names already in the interface:

  Receive(a);(Receive(b)+Receive(a))

  ≤

  Receive(a);Receive(b)

Summary of Results

- Refinement with knowledge about other initial contracts limited to I/O actions
  (enough to guarantee that refinements that extend the interface are included)
  - "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 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

- Global and Local Choreography
- Contract-based service discovery
- A dynamic update mechanism
- Conclusion

Updatable processes/contracts

- 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 a process calculus with
  - updatable parts a[P] and
  - update actions/primitives a{U}, where U is

$$U ::= P \quad | \quad a[U] \quad | \quad U \parallel U \quad | \quad \bullet$$

Example

- Consider the running system:

$$Client[C] \parallel EShop[S] \parallel Bank[Visa]$$

if the following update is performed:

$$\widetilde{Bank}\{NewBank[\bullet \parallel MasterCard]\}$$

the system becomes:

$$Client[C] \parallel EShop[S] \parallel NewBank[Visa \parallel MasterCard]$$

Compliance analysis

- Compliance contract analysis can be used:
  - to detect if several systems correctly interact by composing their behavioural contracts
  - to assess a behavioural contract is internally correct (for complex systems, e.g., cloud)
- Decidability separation results depending on fragments of the language (update power/dynamic topology) [forte-fmoods 2011]

Plan of the Talk

- Global and Local Choreography
- Contract-based service discovery
- A dynamic update mechanism
- Conclusion

Future work

- Contracts with operators for process interruption and compensation
  - The contract language becomes partially undecidable

Related work

- Carbone, Honda, Yoshida
  - Global and End-point calculus similar to our WS-CDL and BPEL4Chor
  - Only some of our observation criteria are considered
  - Stronger conditions for projection

Related work

- Fu, Bultan, Su
  - Service systems with message queues similar to ours
  - Observe the send event as in our sender observation criterion
  - No refinement

Related work

- Padovani et al.
  - Contracts described with an ad-hoc transition system (reminiscent of acceptance tree)
  - The absence of maximal subcontract relation solved either with explicit interfaces of filters (cut the additional actions of the refinements)

Related work

- van der Aalst et al.
  - Contracts described with open workflow nets (similar to petri nets)
  - Same notion of compliance
  - Same definition of subcontract as maximal refinement that preserves compliance
  - Characterization of the refinement for processes without “loops” (make the system infinite due to message queues)

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
- I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the Gap Between Interaction- and Process-oriented Choreographies. In SEFM’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, 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, LNCS, to appear