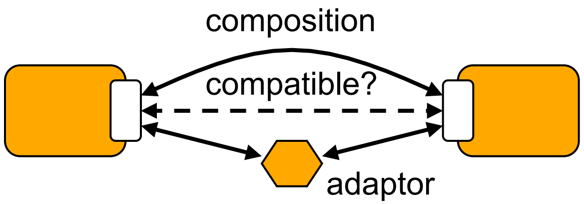
Adaptation of Asynchronously Communicating Software

Carlos Canal
University of Málaga, Spain

Gwen Salaün
Grenoble INP, Inria, France

Software Adaptation

- Direct reuse and composition of existing services is often impossible because their interfaces are incompatible
- Software adaptation aims at automatically generating adaptors enabling non-intrusive composition of black-box services



- All the messages pass through the adaptor which acts as an orchestrator, and makes the services involved work correctly together by compensating for mismatches
- Several levels of interoperability on service interface models: signature, behaviour, semantics, quality of service

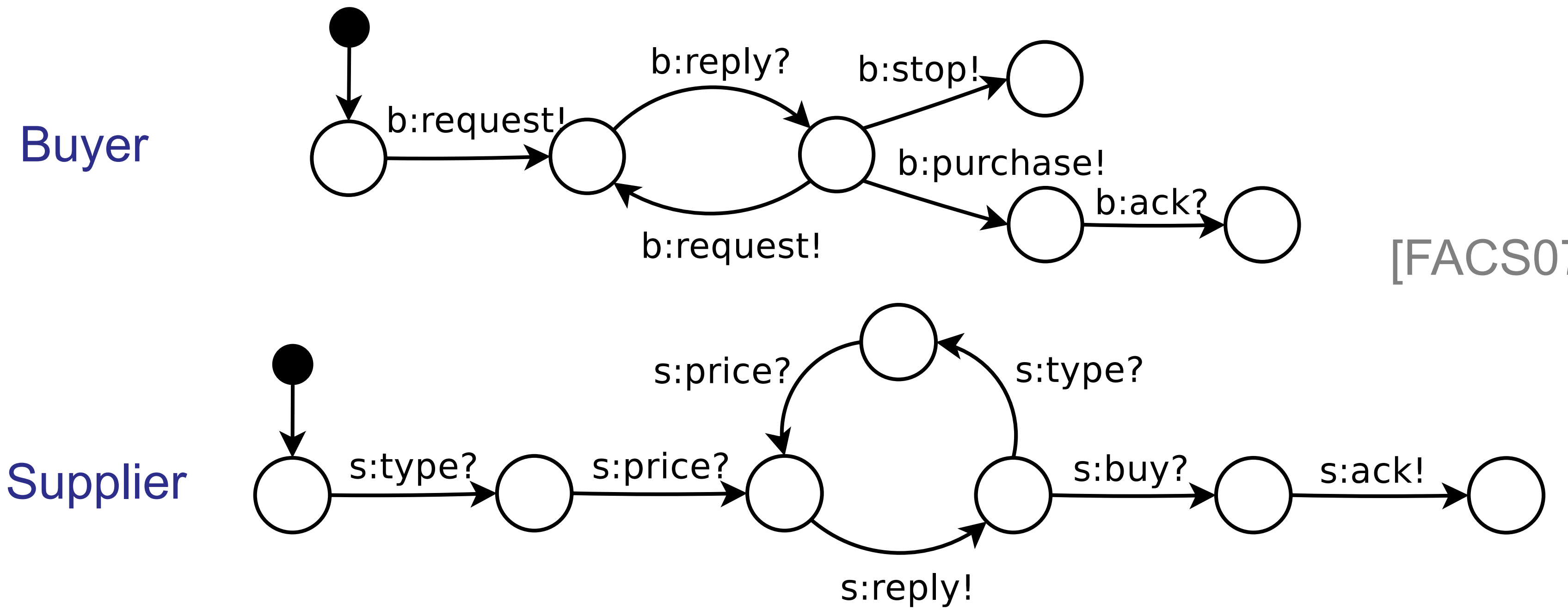
Our Approach

- Most solutions assume that peers interact synchronously (rendez-vous)
- Asynchronous communication (FIFO buffers) is omnipresent but highly complicates the adapter generation process (infinite systems)
- We want to avoid imposing any kind of bounds on buffers, cyclic behaviour, or the number of participants
- Our solution for generating asynchronous adapters combines
  - the synchronizability property for “characterizing” the system behaviour, and
  - synchronous techniques for generating adapters

Outline

1. Synchronous Adaptation
2. Synchronizability
3. Asynchronous Adaptation
4. Concluding Remarks

Models and Mismatch



[FACS07]

- Name mismatch: *purchase! vs buy?*
- Mismatching number of messages: *request! vs type?* and *price?*
- Independent evolution: *stop!*

Adaptation Contract

- Vectors define correspondences between messages
- Adaptation contract for the running example:

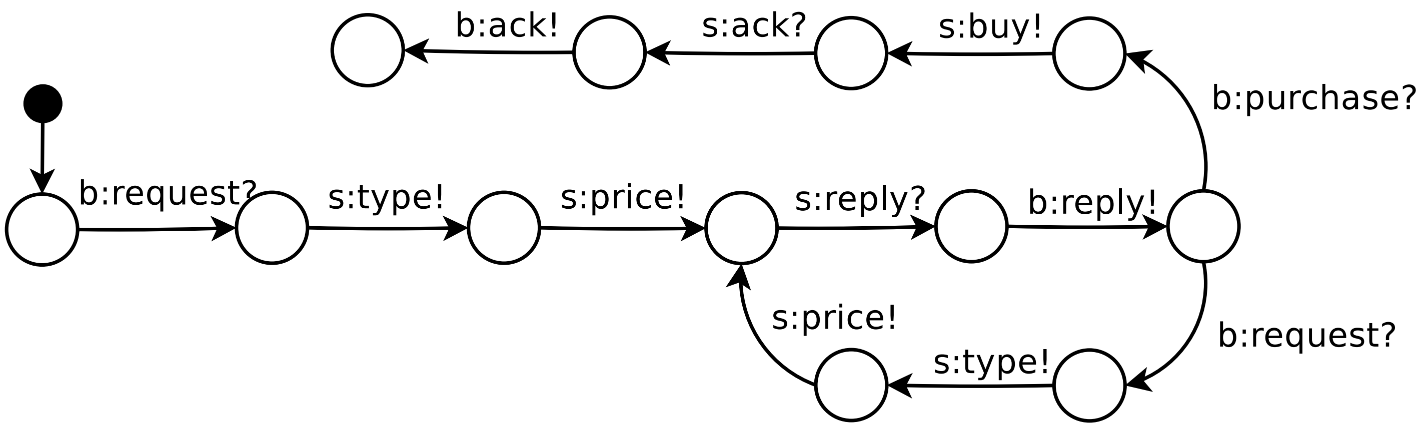
$$V_{req} = \langle b{:}request!,\ s{:}type? \rangle$$
$$V_{price} = \langle b{:}\varepsilon,\ s{:}price? \rangle$$
$$V_{reply} = \langle b{:}reply!,\ s{:}reply? \rangle$$
$$V_{buy} = \langle b{:}purchase!,\ s{:}buy? \rangle$$
$$V_{ack} = \langle b{:}ack?,\ s{:}ack! \rangle$$

where for instance

- V_{buy} solves the name mismatch
- V_{req} and V_{price} solve the mismatching number of messages

Adapter Generation

- Inputs: a set of services LTSs and an adaptation contract
- Output: an adapter LTS (generation of BPEL code possible too)
- Approach: encoding into process algebra and reduction techniques [TSE12]
- Full automation using the Itaca toolset [ICSE09]



Outline

1. Synchronous Adaptation
2. Synchronizability
3. Asynchronous Adaptation
4. Concluding Remarks

Synchronizability

- A set of peers is synchronizable iff the 1-bounded asynchronous system observationally behaves as the synchronous one [POPL12,FACS13]

- If this is the case, the system remains the same for any buffer size:

$$LTS_s \approx_{br} LTS_a^1 \Leftrightarrow \text{forall } k \geq 1 \; LTS_a^k \approx_{br} LTS_a^{k+1}$$

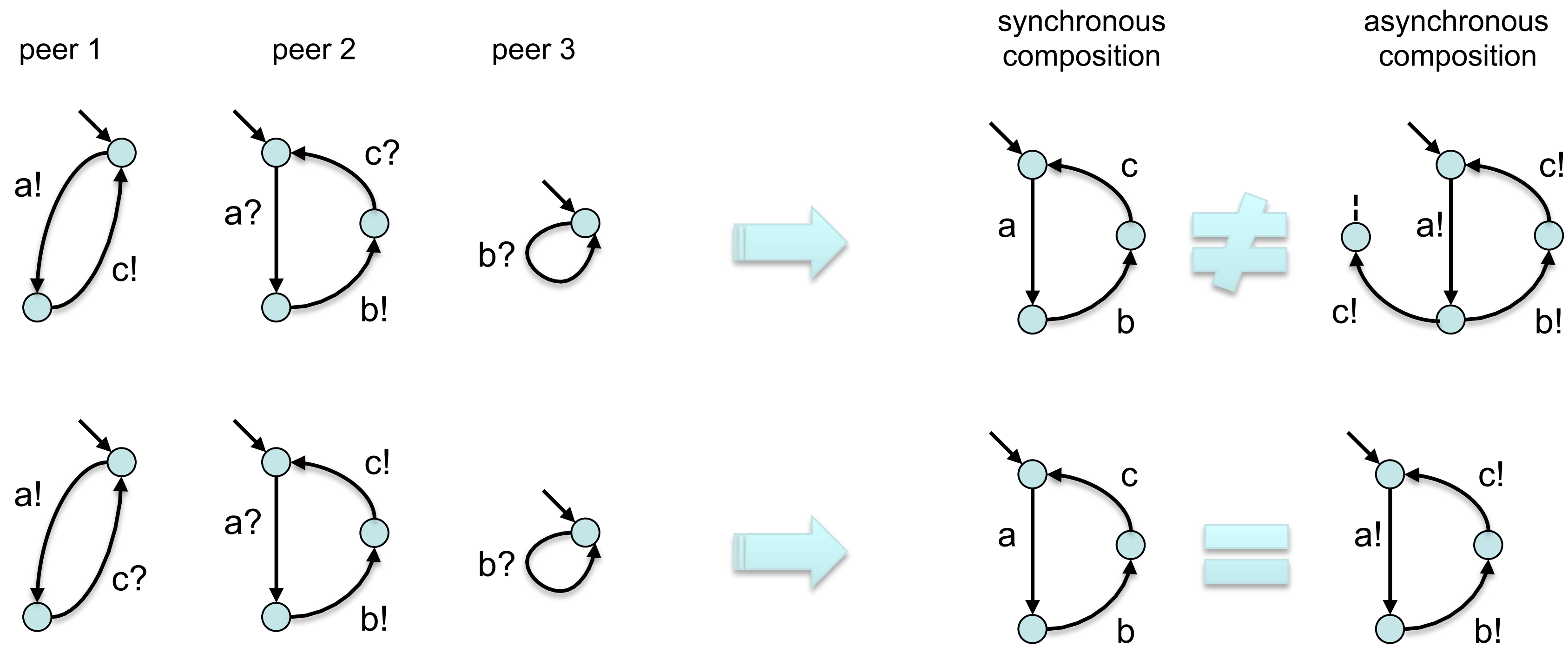
- Synchronizability only considers the ordering of send actions (observable on the network) and ignore the ordering of receive actions (private info.)

- Synchronizability can be verified using equivalence checking techniques

- Synchronizability checking involves finite state spaces, yet the system can be infinite if unbounded (buffer explosion + message consumption)

Well-formedness

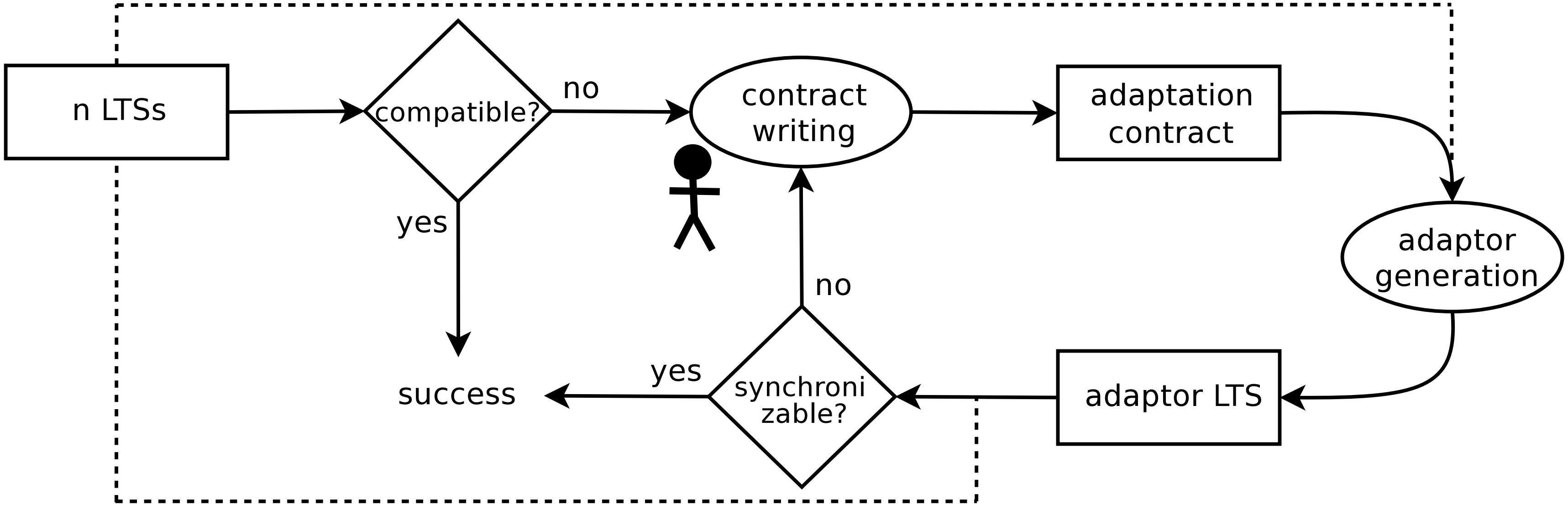
- A set of peers is well-formed iff every send message is eventually received [POPL12,FACS13]
- A synchronizable system consisting of deterministic peers is well-formed



Outline

1. Synchronous Adaptation
2. Synchronizability
3. Asynchronous Adaptation
4. Concluding Remarks

Methodology



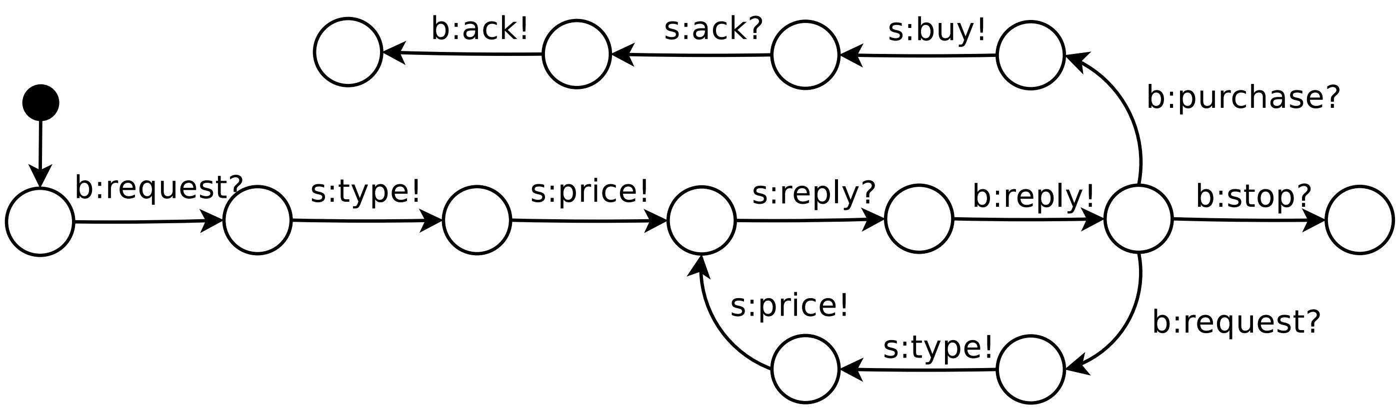
Case Study

- The synchronizability check (peer and adapter LTSs) returns *false*

  b:request!, s:type!, s:price!, s:reply!, b:reply!, and b:stop!

where b:stop! appears in the asynchronous system but not in the synchronous one

- Stop! is not captured by any vector ➡ V_{stop} = <*b:stop!*, *s:ε*>



- The system is synchronizable and this adapter can be used in asynchronous environments

Tool Support

- Itaca toolset for generating synchronous adapters
- Encoding into process algebra and equivalence checking (CADP toolbox) for synchronizability checking

Example	$\|P\|+1$	$\|S\|/\|T\|$	LTS_a^1 ($\|S\|/\|T\|$)	Synchro.	Time
FTP Transfer [4]	3	20/17	13/15	×	52s
Client/Server [10]	3	14/13	8/7	√	54s
Mars Explorer [6]	3	34/34	19/22	×	49s
Online Computer Sale [13]	3	26/26	11/12	√	53s
E-museum [11]	3	33/40	47/111	×	53s
Client/Supplier [8]	3	31/33	17/19	√	49s
Restaurant service [29]	3	15/16	10/12	√	55s
Travel Agency [27]	3	32/38	18/21	√	52s
Vending Machine [16]	3	15/14	8/8	√	49s
Client/Server [28]	4	19/24	18/32	×	64s
SQL Server [26]	4	32/38	20/27	×	62s
Booking System [20]	5	45/53	27/35	×	85s

Outline

1. Synchronous Adaptation
2. Synchronizability
3. Asynchronous Adaptation
4. Concluding Remarks

Concluding Remarks

- Most existing approaches assume synchronous communication for generating adapters
- Our approach combines synchronous adaptation techniques and the synchronizability property for iteratively generating asynchronous adapters
- Our solution is fully supported by several tools
- Main perspective: avoiding the iterative approach, *e.g.*, by guiding the designer to build synchronizable systems by construction