Implementability of Message Sequence Charts

F. Khendek, G. Robert, G. Butler and P.Grogono

Concordia University

Montreal, Canada

khendek@ece.concordia.ca

Plan

- Why ?
- Basic Algorithm
- Extensions
- Problems:
  - Implementability issue
  - Compatibility between MSCs
- Discussion

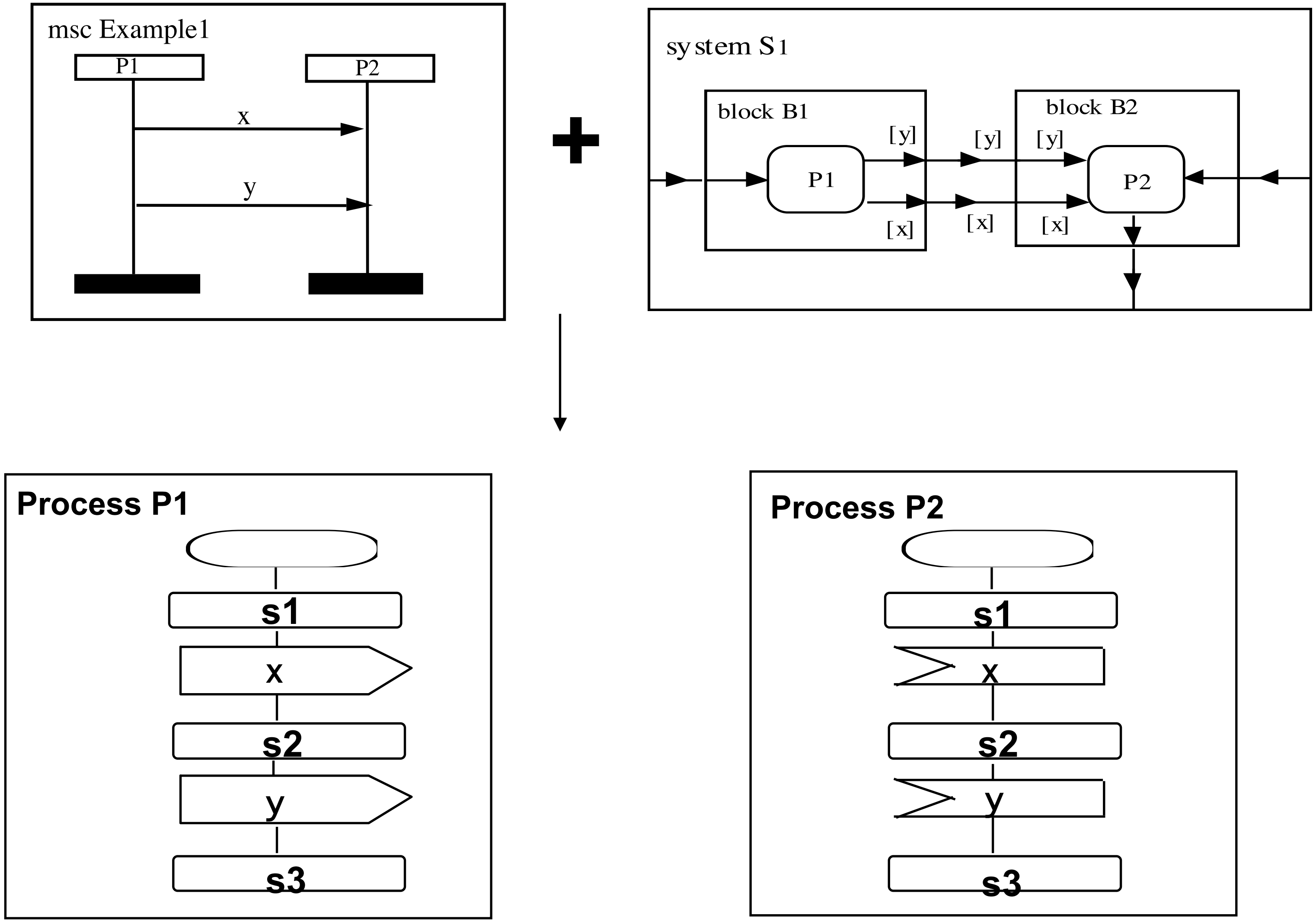
Why ?

- **From requirements to design specification (at least for the <u>behavioral aspect</u>): ensure consistency by construction**
- **Incremental design of SDL specifications (Add traces in a stepwise manner)**

$$\begin{tikzcd} \mathrm{MSC}_0 \arrow[r] & \mathrm{SDL}_0 + \mathrm{MSC}_1 \arrow[r] & \mathrm{SDL}_1 + \mathrm{MSC}_2 \arrow[r] & \mathrm{SDL}_2 \ \ldots \end{tikzcd}$$

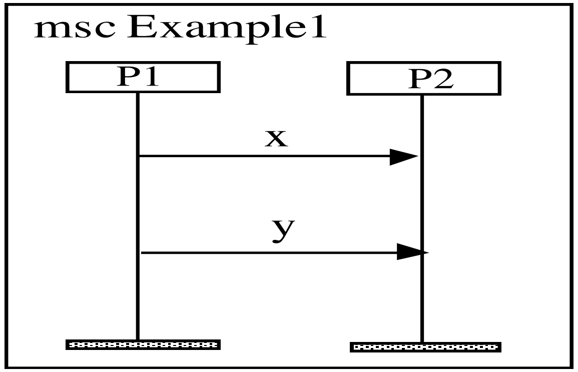
- **Enrich existing SDL specification without modifying the architecture → adding services (“service creation”)**

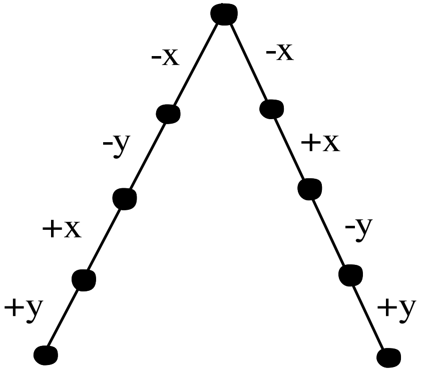
Basic Approach: Introduction

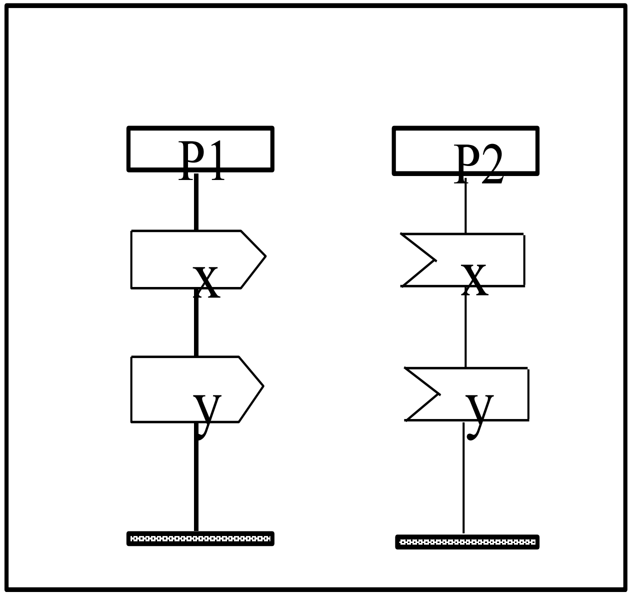


Basic Approach: issues

- MSC specifies required order of sending and consumption of messages







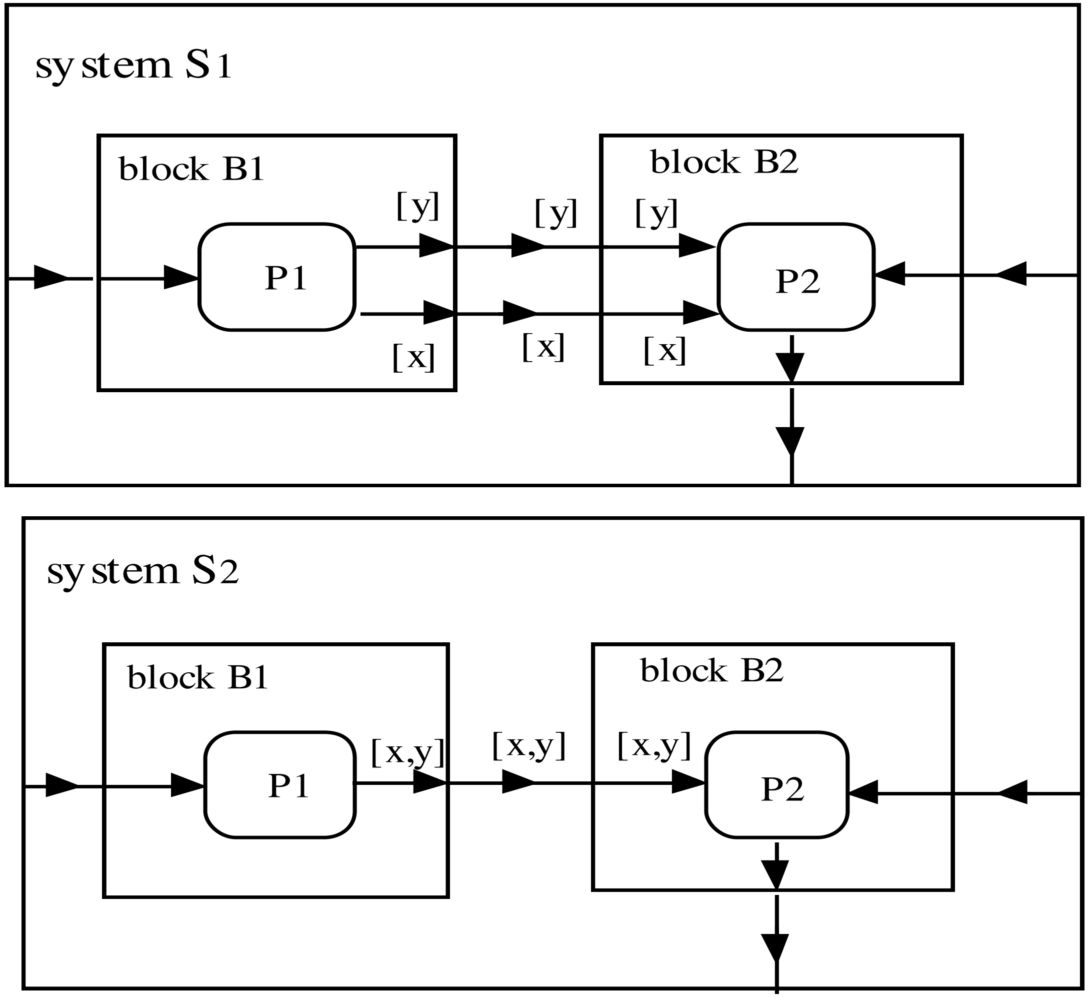
Translation seems straightforward !

However …

Basic Approach: issues

- DOES NOT specify how process instances communicate
- The actual arrival depends on the communication architecture
- The given SDL architecture defines the communication architecture
- Even with a defined communication architecture the actual arrival of messages (signals) into SDL process instance queue may be different from the consumption order
- Straightforward translation may lead to deadlocks because of SDL implicit transitions...

Basic Approach: issues



Two different architectures

Basic Approach: Key Concepts

- For each process, generate an SDL skeleton with the sending and receiving transitions as specified in the bMSC, BUT keep in mind all the possible arrival orders to the input queue **<u>according to the given architecture</u>**
- Avoid implicit transition for signals that will be consumed later
- APPROPRIATE USE of "SAVE": If process instance is expecting signal *y*, then "any" other signal that MAY BE in the queue and ahead of *y* is saved

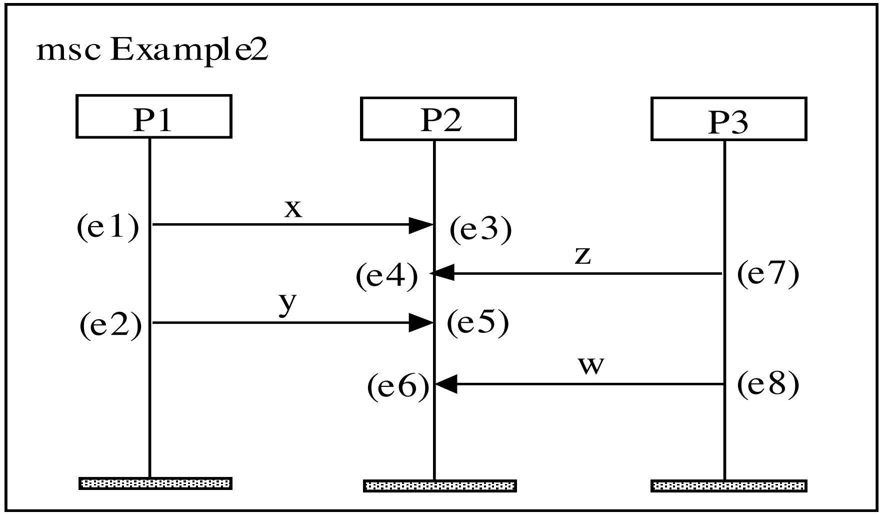
- 3 Main steps in the translation algorithm

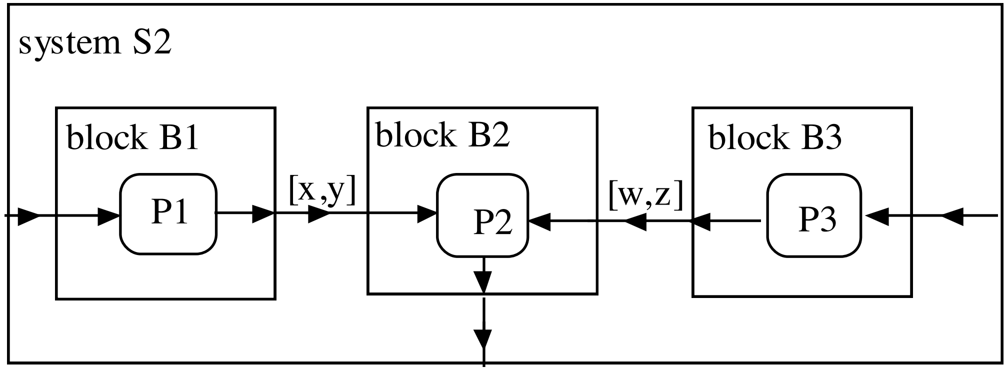
Basic Approach: Step 1

- First step: Ordering of events *
  - define a transitive *earlier* relation << , $e_i << e_j$ means e_i occurs *earlier* in time than e_j
  - two rules:
    - for each MSC instance, events are totally ordered
    - the sending event of a message occurs *earlier* than its reception
  - Transitive closure of the order relation is independent from the architecture

*** Similar to Holzman and Alur et al. in their work on race conditions**

Example: Step 1





	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
e_1		T	T	T	T	T		
e_2					T	T		
e_3				T	T	T		
e_4					T	T		
e_5						T		
e_6								
e_7				T	T	T		T
e_8						T		

(e_i, e_j) = T means $e_i << e_j$

Basic Approach: Step 2

- Build “receive queues”
  - For each process, in order to view the possible arrival orders of incoming signals, we view its input queue as a set of parallel FIFO queues. Each queue correspond to one incoming channel
  - Algorithm creates a table for each process:
    - 1 column for each “receive queue” (for each incoming channel)
    - a row for each input event (and only input events)
    - for each instance P_i in the MSC
      - for each output event e_s sending signal m to P_j
        - » find the related input event e_r in P_j
        - » for each input event e_k in instance P_j

          if not($e_k << e_s$) and not($e_r << e_k$),

          add signal m to the appropriate “receive queue”

Example: Step 2

Event	Input Signal	$Q_{1,2,1}$	$Q_{3,2,1}$
e_3	x	x,y	z,w
e_4	z	y	z,w
e_5	y	y	w
e_6	w		w

“Receive queues” table for process P_2

Basic Approach: Step 3

- Generate SDL code (use of SAVE)
- for each instance P_i in the MSC diagram
  - for each event e_j
  - if e_j is an output event generate an SDL output
  - else if e_j is an input event of signal m
    - generate an SDL input for message m
    - for each "receive queue" of P_i (except the queue to which m belongs), generate an SDL SAVE for all the messages in the queue

      [THESE MESSAGES MAY ARRIVE INTO P_I INPUT QUEUE BEFORE m]

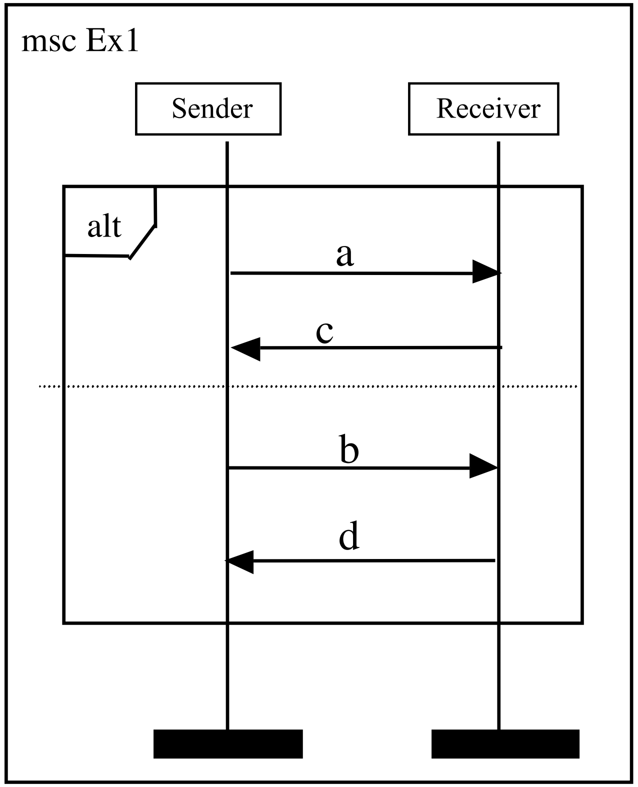
Example: Step 3

SDL specification of process P_2

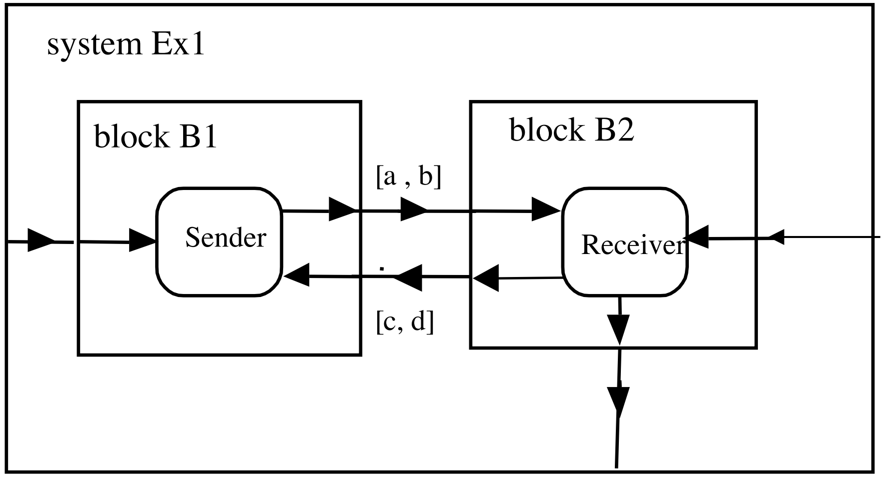
Extensions

- Inline constructs:
  - alt
  - opt
  - seq,
  - loop, etc.

Extensions: Alt construct

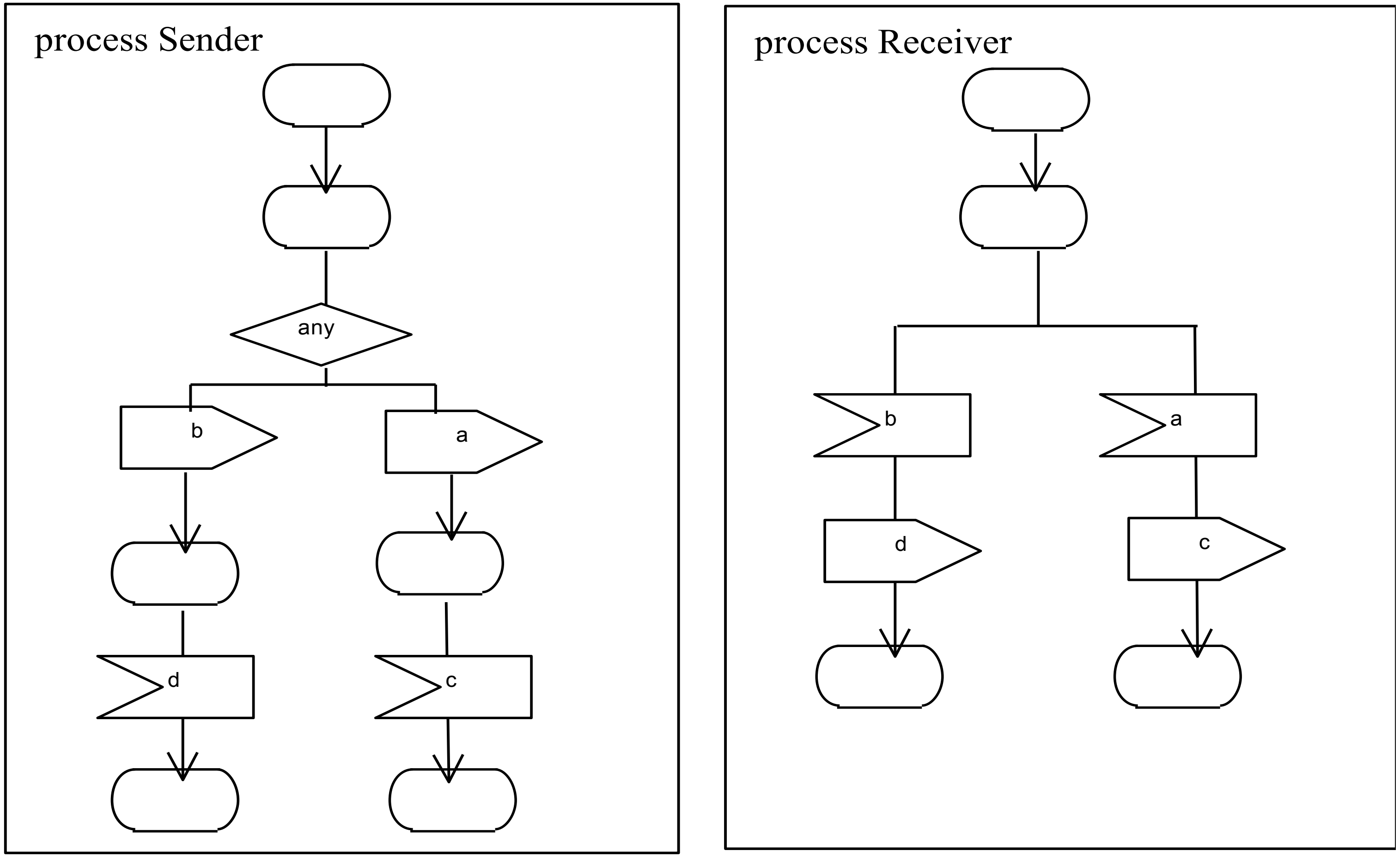


Example 1

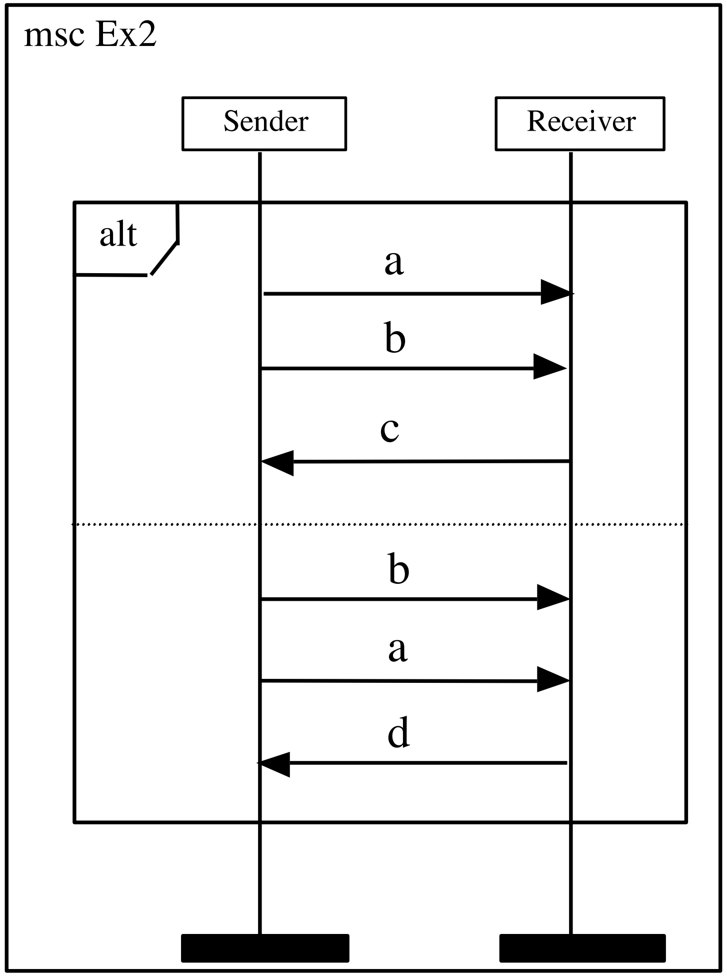


Extensions: Alt Construct

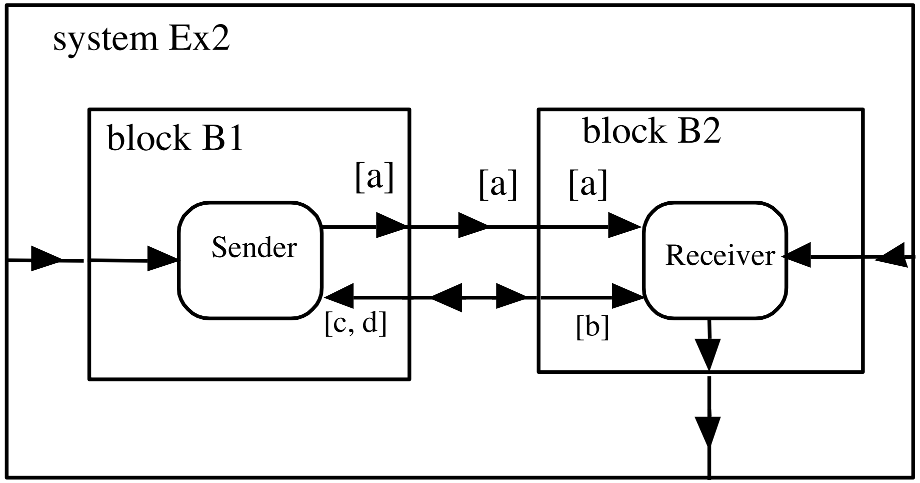
Generated SDL processes for Example 1



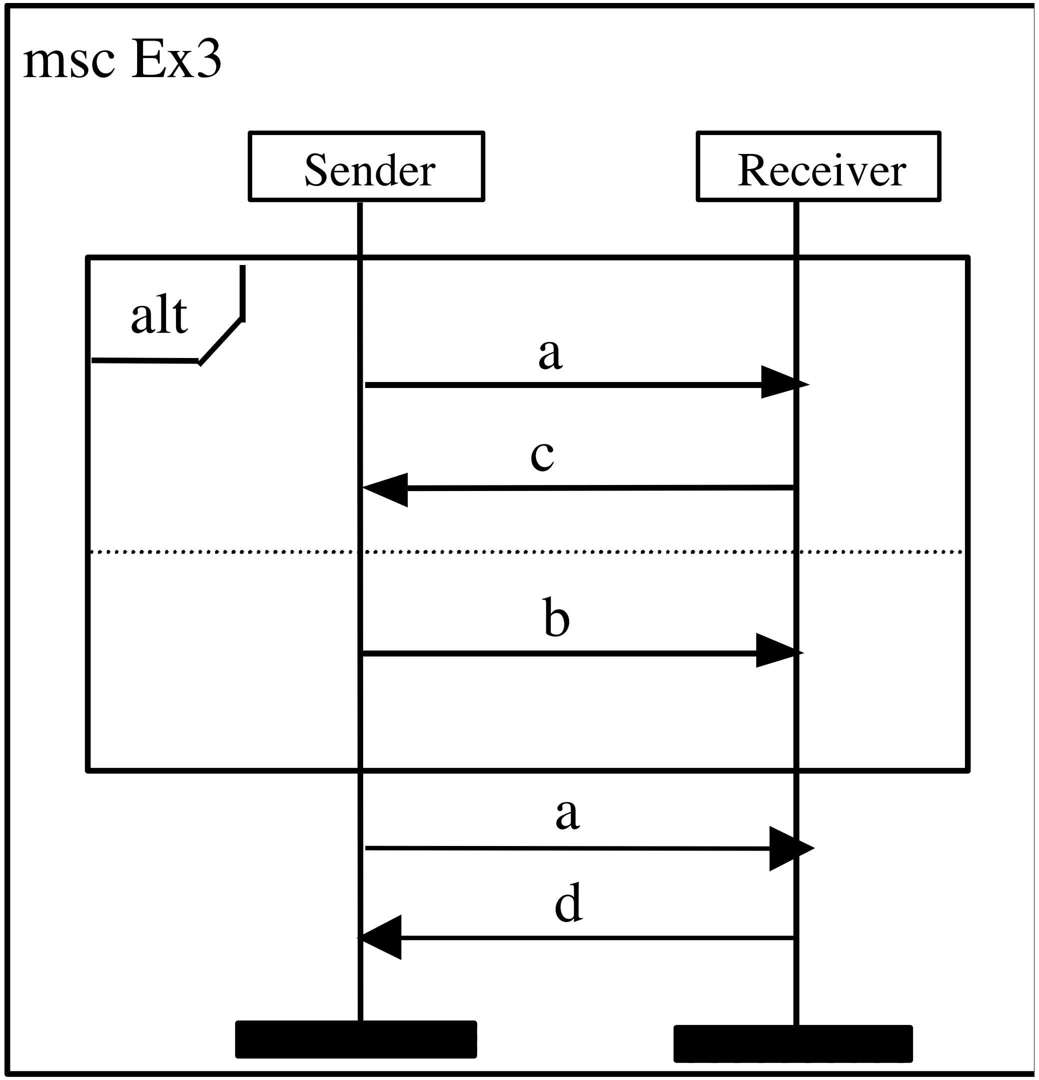
Extensions: Alt Construct



Example 2: **Problems !**



Extensions: A Second alt Example

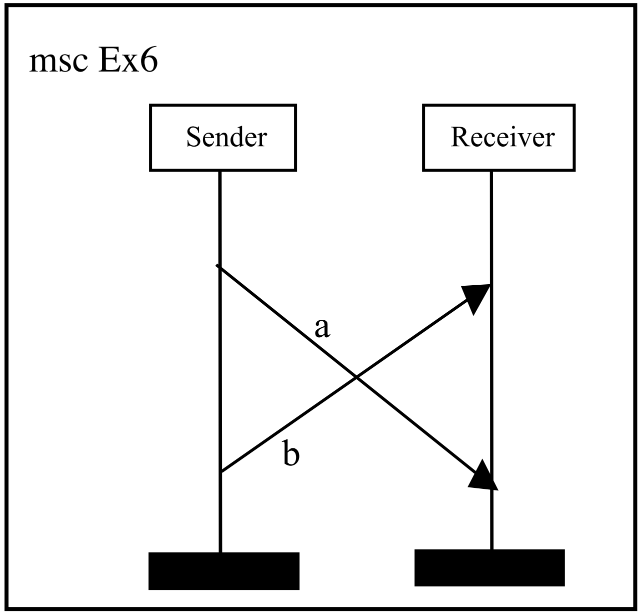


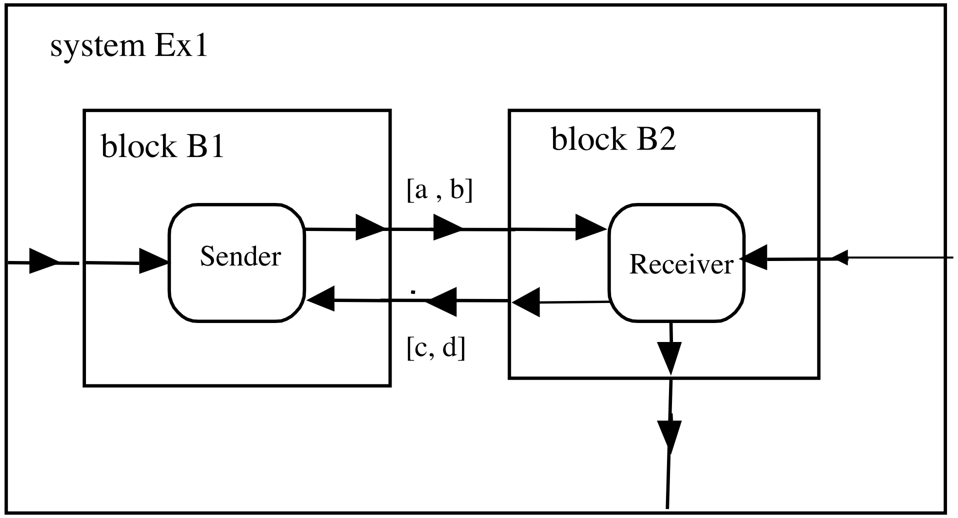
Example 3:

Problems !

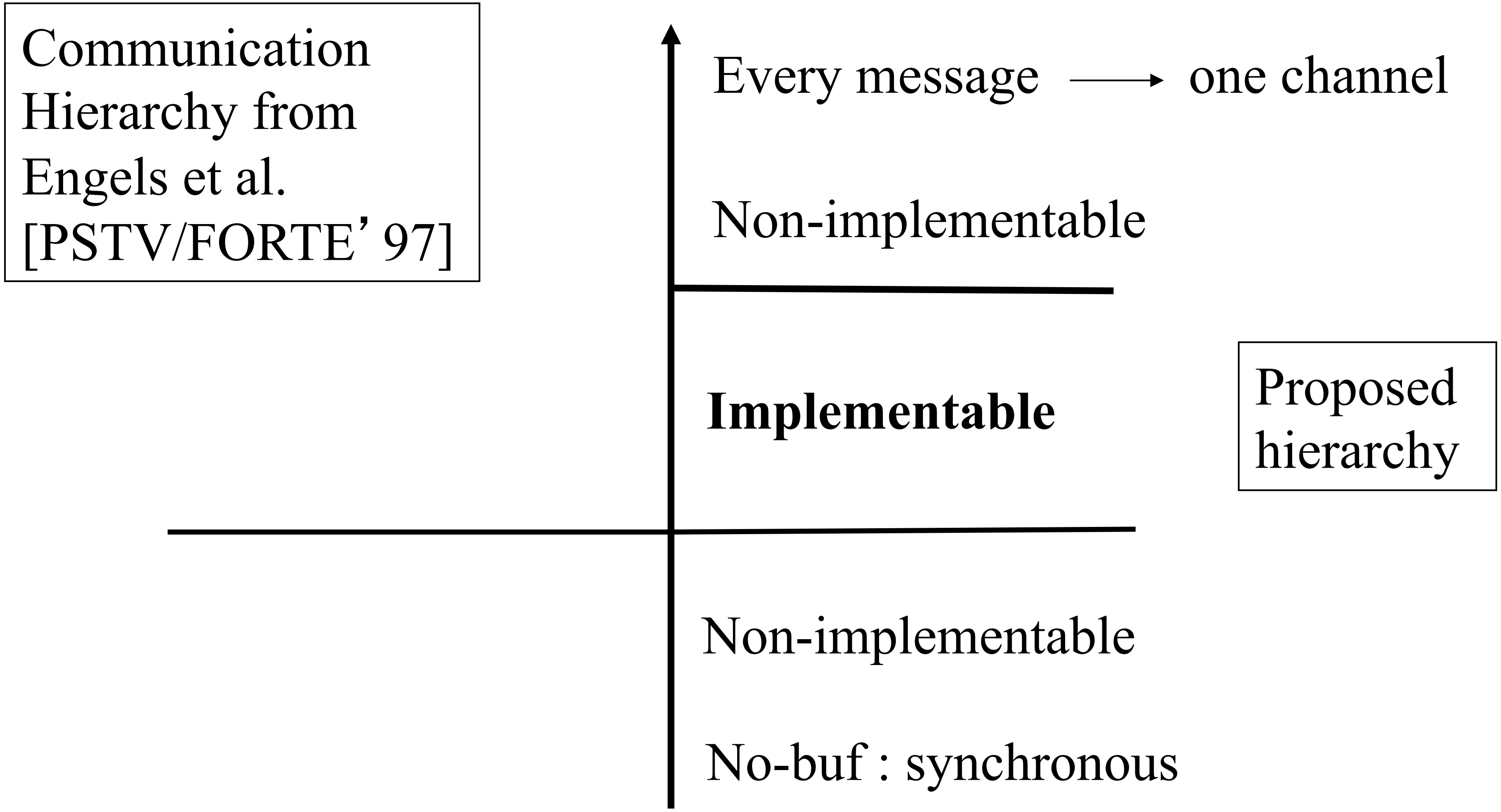
Extension: Overtaking

Example 4: **Problems !**





Communication hierarchy



Communication Hierarchy from Engels et al. [PSTV/FORTE’ 97]

Every message ⟶ one channel

Non-implementable

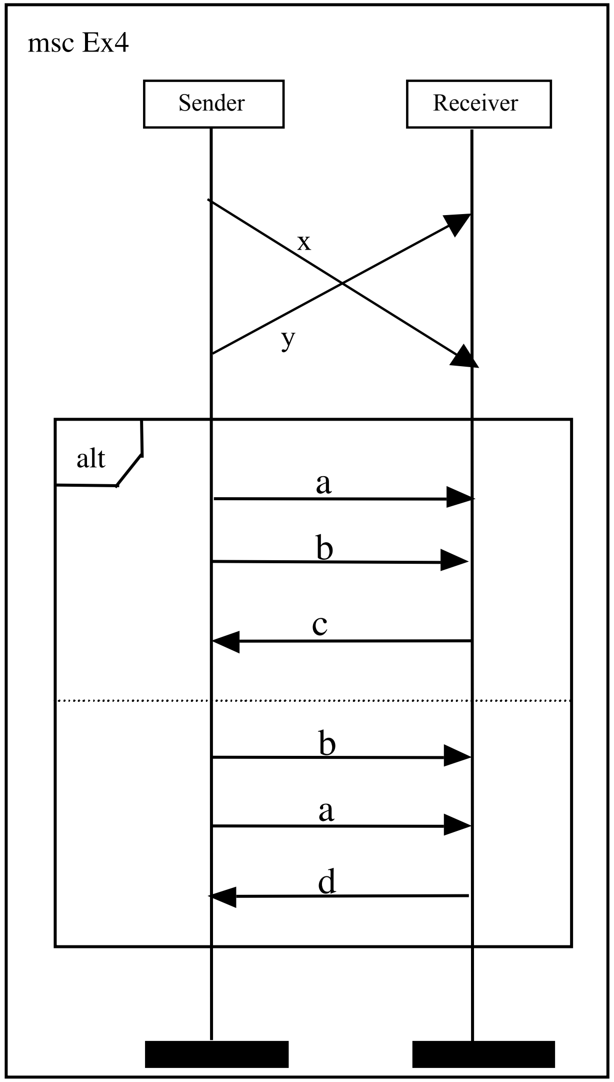
Implementable

Non-implementable

No-buf : synchronous

Proposed hierarchy

Communication hierarchy (cont.)



Example 5

Cannot be implemented with full synchronization or msg-models.

Compatibility between MSCs

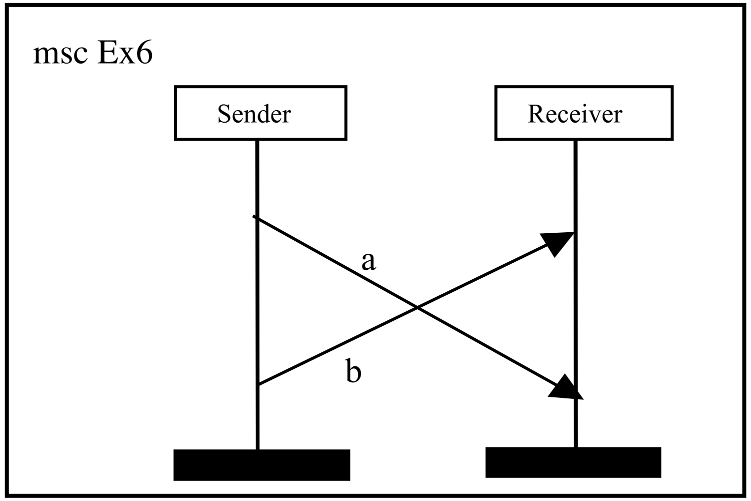
- Related to implementability

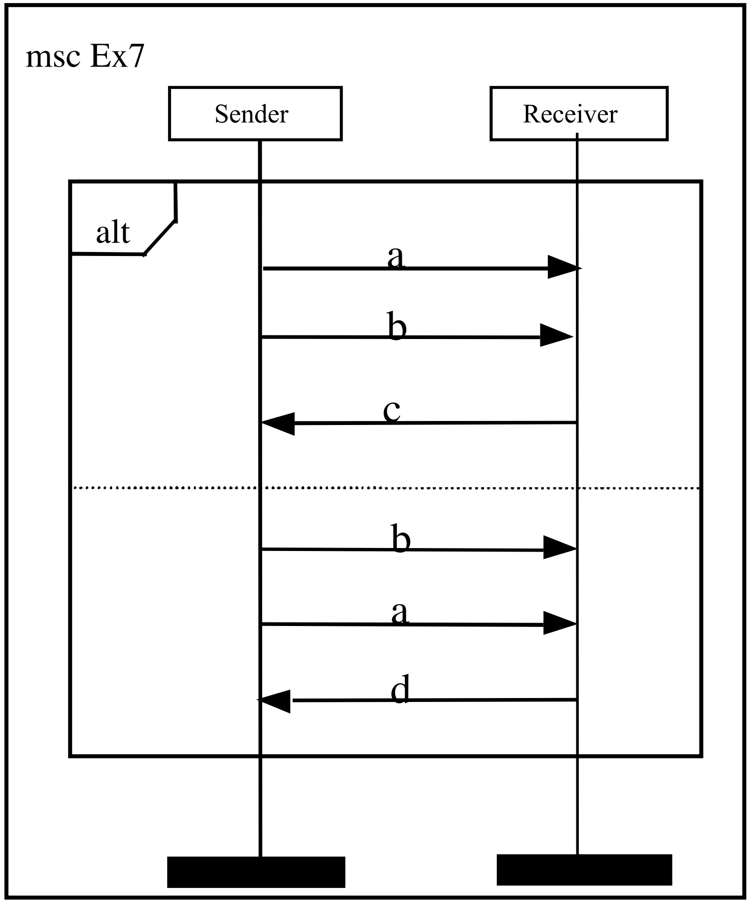
- Two MSCs are compatible, if they can be implemented in the same architecture.

- MSC Composition Operators ?

Compatibility between MSCs (cont.)

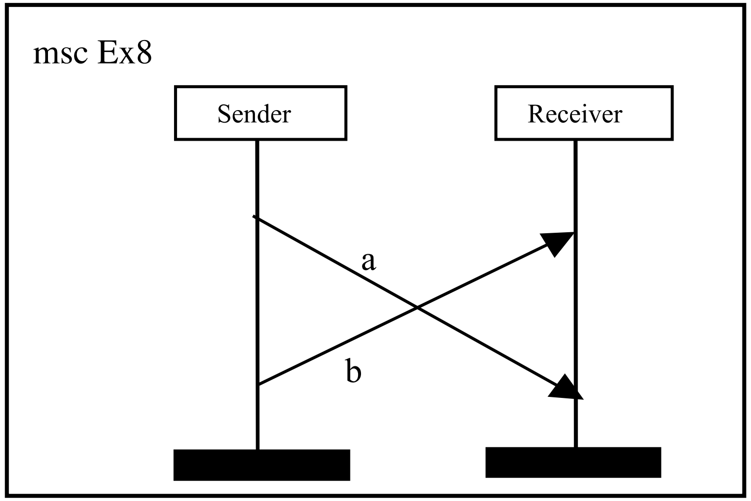
These two MSCs are incompatible.

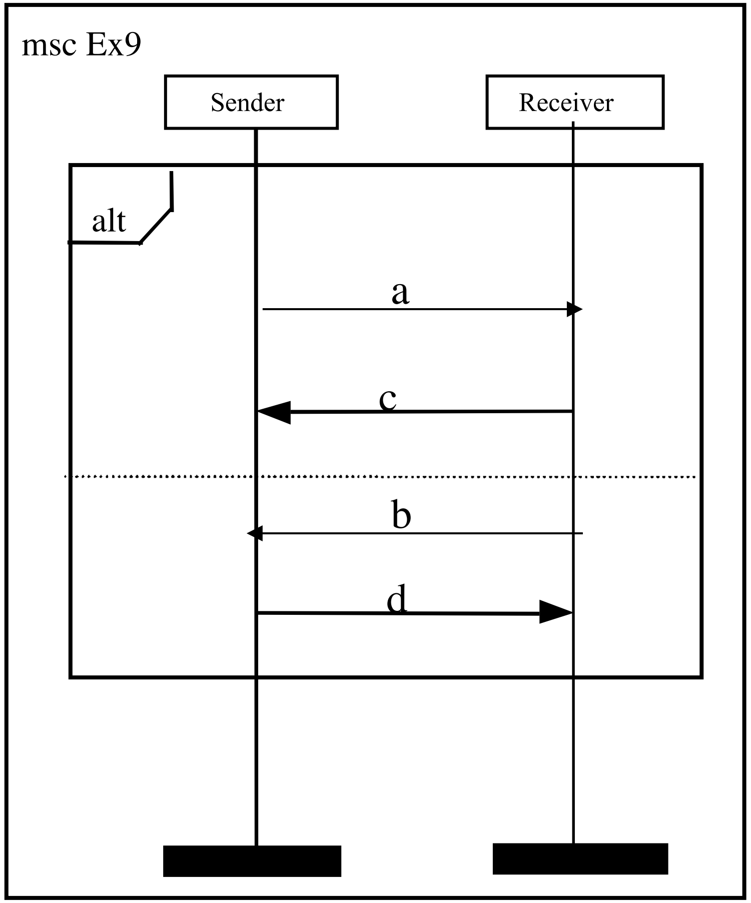




Compatibility between MSCs (cont.)

These two MSCs are incompatible.





Discussion

- Different issues simultaneously: translation, Implementability, compatibility
- Data part ?
- Environment for enriching SDL specifications : use ObjectGeode Internal Representation
- A basis for maintaining code ...
- Work is still in progress ...