SRML

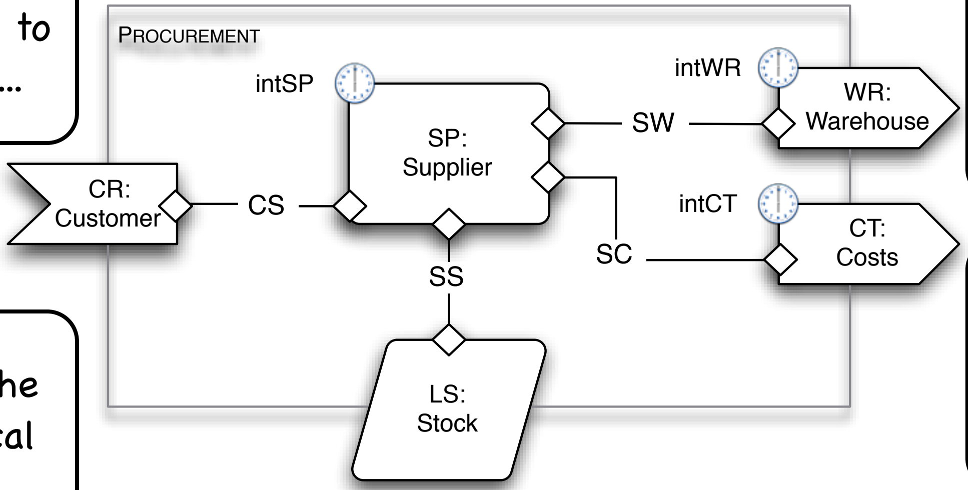
primitives

Laura Bocchi
bocchi@mcs.le.ac.uk

The service Procurement

Procurement service … provides a service to a customer that wants to purchase a product…

…first we check if the product is in the local stock…



…if the product is not in stock searches for a warehouse that can provide the product…

…we use a service to quote the specific product..

Specification Languages

Business Roles	Business Protocols	Layer Protocols	Interaction Protocols	
=	=	=	=	
Interactions	Interactions	Interactions	Role A	Role B
+	+	+	+	
Orchestration	Behaviour	Behaviour	Coordination	

Declaring interactions (1/2)

- an asynchronous interaction is defined by
  - interaction type
  - interaction name
  - parameters
- parameters are defined by
  - associated event
  - parameter name
  - parameters type

```
BUSINESS ROLE Supplier is
  INTERACTIONS
  r&s requestQuote
    🔔    which:product
    ✉    cost:money
  r&s orderGoods
    🔔    many:nat
    ✉    much:money
  rcv makePayment
  snd shipOrder
  s&r checkShipAvail
     🔔 which:product, many:nat
  rcv confirmShip
  ask how(product):money
  ask checkStock(product,nat):bool
  tll incStock(product,nat)
  tll decStock(product,nat)
```

Declaring interactions (2/2)

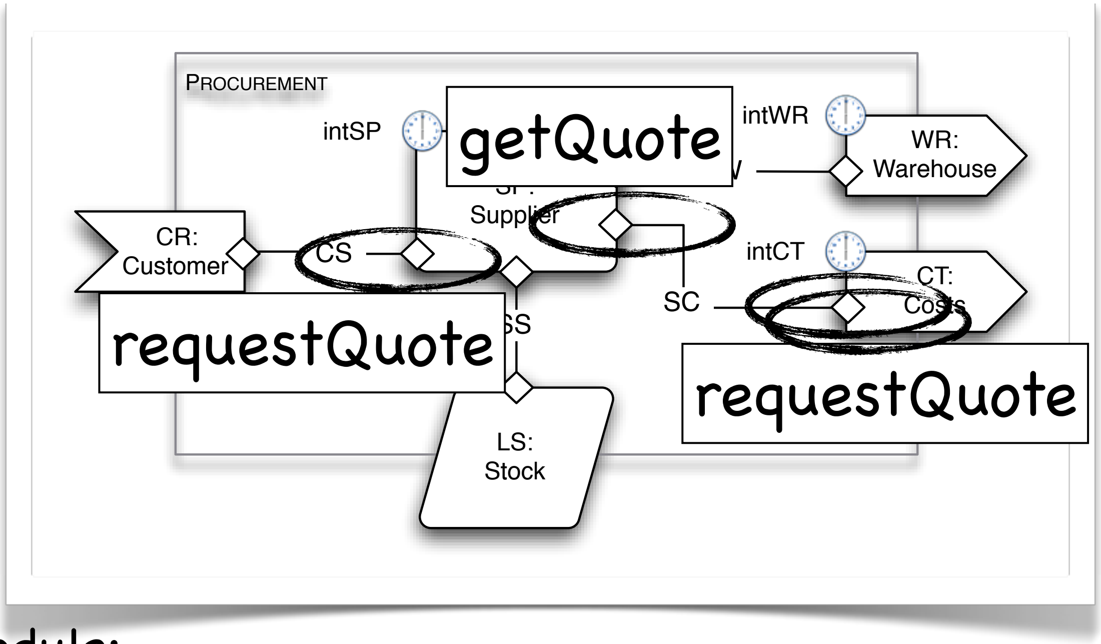
- a synchronous interaction is defined by
  - interaction type
  - interaction name
  - input types
  - output types

```
BUSINESS ROLE Supplier is
  INTERACTIONS
  r&s requestQuote
   🔔   which:product
   ✉   cost:money
  r&s orderGoods
   🔔   many:nat
   ✉   much:money
  rcv makePayment
  snd shipOrder
  s&r checkShipAvail
    🔔 which:product, many:nat
  rcv confirmShip
  ask how(product):money
  ask checkStock(product,nat):bool
  tll incStock(product,nat)
  tll decStock(product,nat)
```

Interaction Names

- Each node (component interface, EX-P, EX-R, uses/serves-interface) has a type which is its specification
- Each specification declares a set of interactions
- Each specification identifies each interaction through a name which is unique for that specification
- Each specification has been defined, maybe, independently (e.g., in different times and places)

Interaction Names



- In a module:
  - two nodes may be instances of specifications that use the same name for pairs of interaction that are unrelated in the module
  - two communicating nodes may be instances of specifications that use different names for pairs of interactions that are related
- The “coupling” of interactions is done explicitly with the wires

Synchronous Interaction Types

- The sender blocks while waiting for the reply
  - synchronisation on performing an operation

    e.g., incStock(product,nat)
    - tll the party requests the co-party to perform an operation and blocks
    - prf party performs an operation and frees the co-party that requested it
  - sychronisation with data transfer

    e.g., checkStock(product,nat):bool
    - ask ask the party synchronizes to obtain data
    - rpl the party synchronizes to transmit data

Asynchronous Interaction Types

- The sender does not block waiting for the message to be received
  - One-way: only involve one event
    - snd the interaction is initiated by the party
    - rcv the interaction is initiated by the co-party
  - Conversational: start a conversation involving multiple **events**
    - s&r the conversation is initiated by the party
    - r&s the conversation is initiated by the co-party

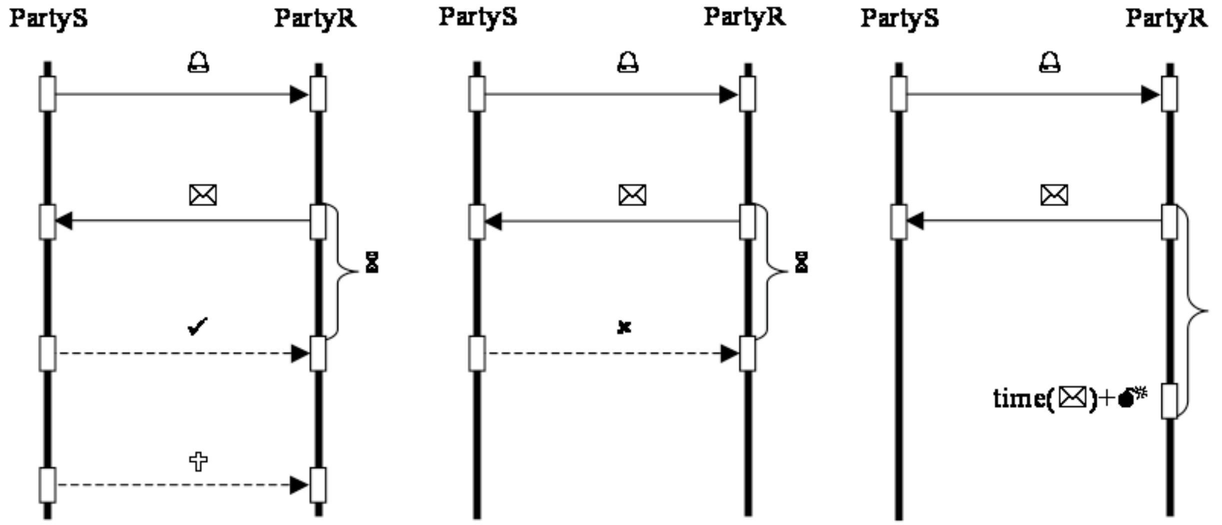
Event Types

- One-way interactions are associated ONLY to initiation events (i.e., 🔔-events)
- Conversational interactions can be associated to a number of interaction events:

`interaction🔔`	The event of initiating *interaction*.
`interaction✉`	The reply-event of *interaction*.
`interaction✓`	The commit-event of *interaction*.
`interaction×`	The cancel-event of *interaction*.
`interaction✝`	The revoke-event of *interaction*.

Conversations

- **PartyS** declares an interaction **e1** of type **s&r**
- **PartyR** declares an interaction **e2** of type **r&s** (connected via wires to **e1**)
- **PartyS** starts the conversation issuing the first interaction event associated to an interaction name
- A number of events can be associated to an interaction name, corresponding to the different phases of the conversation



Computational model

- **PartyS** declares an interaction **e1** of type s&r and **PartyR** declares an interaction **e2** of type r&s (connected via wires to **e1**)

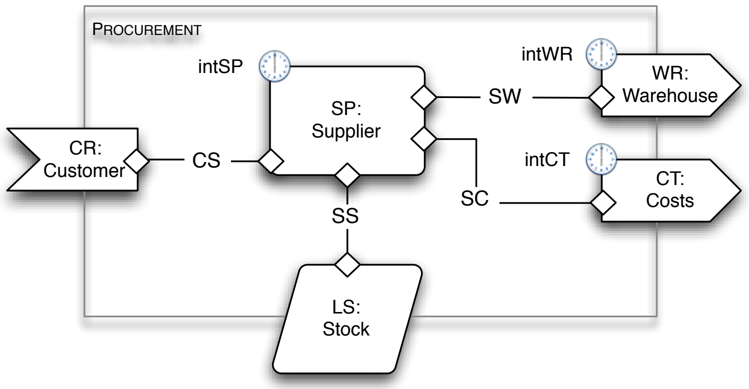
- the **initiation** event for **e** is:
  - (1) issued by the **partyS**: **e🔔!**
  - (2) stored in a buffer of **partyR**,
  - (3) processed by **partyR** and then
  - (4) either executed **e🔔?** by **partyR** or discarded

the same for cancel/commit/revoke events

- the **reply** event for **e** is:
  - (1) issued by the **partyR**: **e✉!**
  - (2) stored in a buffer of **partyS**,
  - (3) processed by **partyS** and then
  - (4) either executed **e✉?** by **partyS** or discarded

Events in SP: Examples

As exercise, we informally describe a fragment of the orchesration of SP in terms of interaction events



```
BUSINESS ROLE Supplier is
  INTERACTIONS
  r&s requestQuote
   🔔   which:product
   ✉   cost:money
  r&s orderGoods
   🔔   many:nat
   ✉   much:money
  rcv makePayment
  snd shipOrder
  s&r checkShipAvail
    🔔 which:product, many:nat
  rcv confirmShip
  ask how(product):money
  ask checkStock(product,nat):bool
  tll incStock(product,nat)
  tll decStock(product,nat)
```

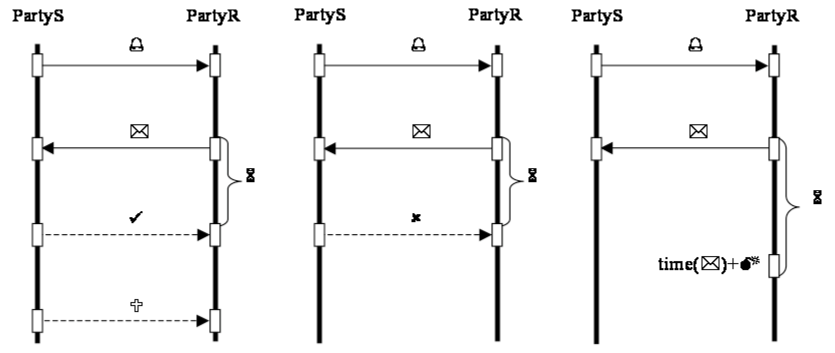
- orderGoods🔔?
- checkStock(requestQuote.which,orderGoods.many) if the product is not in stock SP interacts with WR
  - checkShipAvail🔔!
    - checkShipAvail.which=requestQuote.which
    - checkschipAvail.many=orderGoods.many

Important details (1/3)

- We assume the existence of some environment functions that return (synchronously) information about the time:
  - “today” returns the current date (a value of type “date”)
  - “now” returns the current instant (a value ot type “time”)

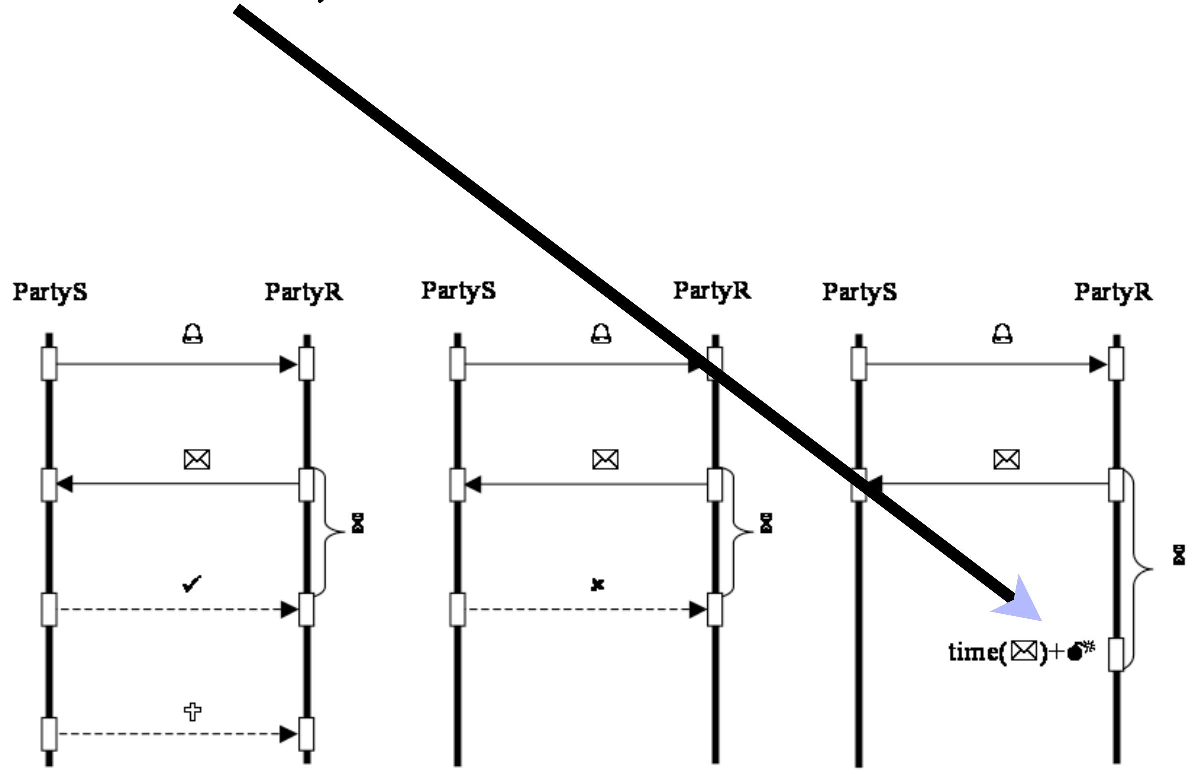
Important details (2/3)

- each reply-event ✉ has two default parameters (i.e., they are defined even if they do not appear in the declaration of the interactions)
  - Reply: is a boolean
  - UseBy: is a value of type time
- If the value of Reply is true, PartyR ensures a number of properties for an interval of time denoted by 💣. Also, the confirm-event and the cancel-event are enabled.
- If the value of Reply is false, no property is ensured and the confirm-event and the cancel-event are not enabled.
- We use the notation interactionName.Reply to denote interactionName.Reply=true and ¬interactionName.Reply to denote interactionName.Reply=false

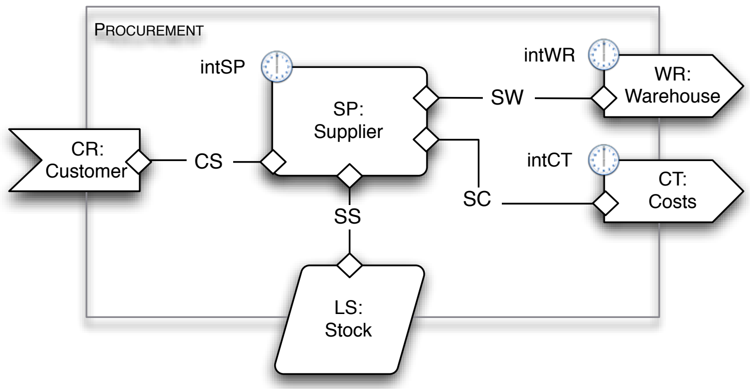


Important details (3/3)

- If the value Reply is true, the parameter UseBy represents the deadline (i.e., the instant from which the properties are not anymore ensured).
- PartyR calculates the value UseBy by adding the interval 💣 to the value now (referring to when the ✉-event is sent)



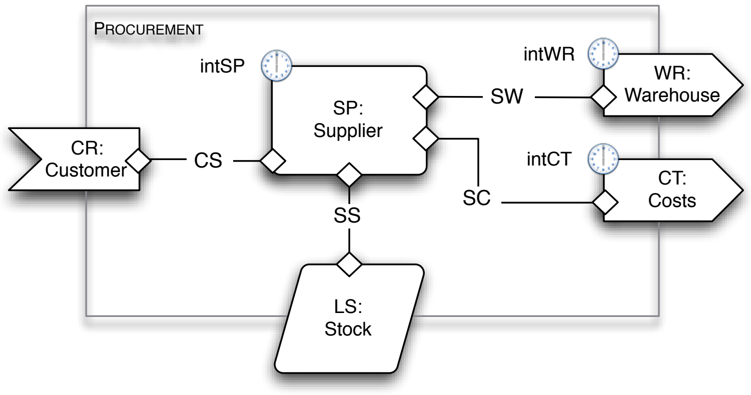
Events in SP: Examples



```
BUSINESS ROLE Supplier is
  INTERACTIONS
  r&s requestQuote
    🔔  which:product
    ✉   cost:money
  r&s orderGoods
    🔔  many:nat
    ✉   much:money
  rcv makePayment
  snd shipOrder
  s&r checkShipAvail
     🔔 which:product, many:nat
  rcv confirmShip
  ask how(product):money
  ask checkStock(product,nat):bool
  tll incStock(product,nat)
  tll decStock(product,nat)
```

- if checkShipAvail✉? and checkShipAvail.Reply=true
- orderGoods✉!
  - the price is fixed for the interval orderGoods💣,
  - orderGoods.Reply is set to true,
  - orderGoods.UseBy is set to now+orderGoods💣,
  - the following events are enabled:
    - orderGoods✓?
    - orderGoods✗? …

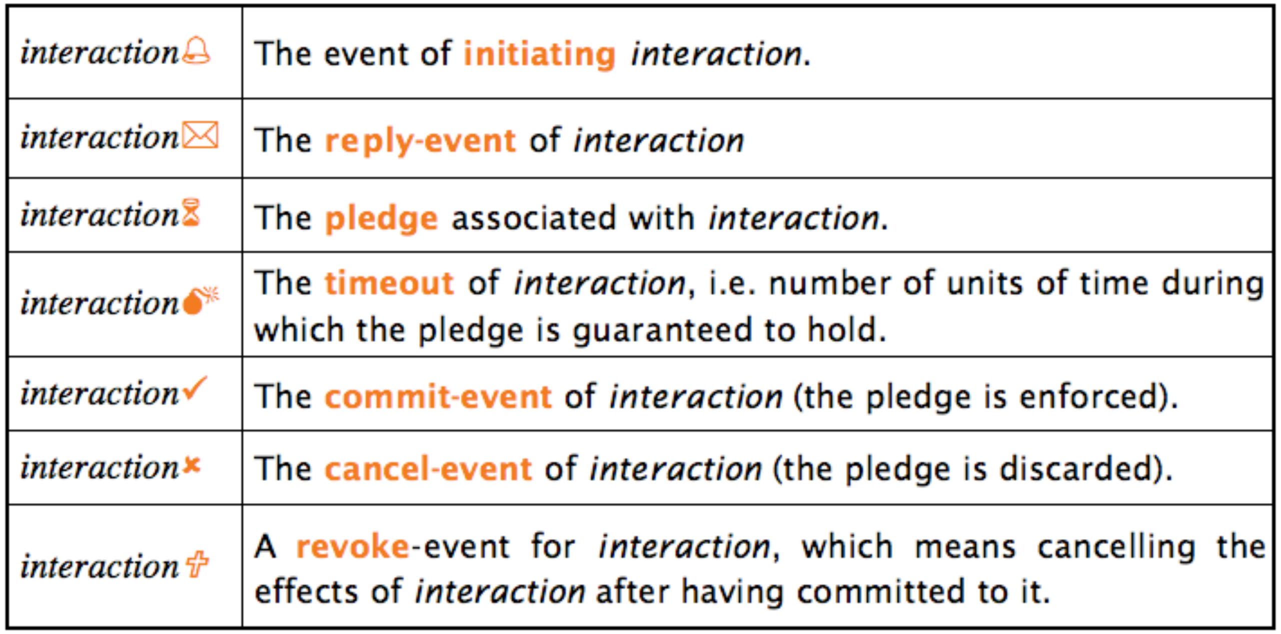
Events in SP: Examples



- if checkShipAvail✉? and checkShipAvail.Reply=false
- orderGoods✉!
  - orderGoods.Reply is set to false

```
BUSINESS ROLE Supplier is
  INTERACTIONS
  r&s requestQuote
   🔔    which:product
   ✉    cost:money
  r&s orderGoods
   🔔    many:nat
   ✉    much:money
  rcv makePayment
  snd shipOrder
  s&r checkShipAvail
    🔔 which:product, many:nat
  rcv confirmShip
  ask how(product):money
  ask checkStock(product,nat):bool
  tll incStock(product,nat)
  tll decStock(product,nat)
```

Iconography of SRML



interaction🔔	The event of **initiating** *interaction*.
interaction✉	The **reply-event** of *interaction*
interaction⌛	The **pledge** associated with *interaction*.
interaction💣	The **timeout** of *interaction*, i.e. number of units of time during which the pledge is guaranteed to hold.
interaction✓	The **commit-event** of *interaction* (the pledge is enforced).
interaction✗	The **cancel-event** of *interaction* (the pledge is discarded).
interaction✝	A **revoke**-event for *interaction*, which means cancelling the effects of *interaction* after having committed to it.