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...

...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..

...first we check if the product is in the local stock...
# Specification Languages

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactions</td>
<td>Interactions</td>
<td>Interactions</td>
<td>Role A</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>Role B</td>
</tr>
<tr>
<td>Orchestration</td>
<td>Behaviour</td>
<td>Behaviour</td>
<td>Coordination</td>
</tr>
</tbody>
</table>
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

spd 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)
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)
  - till 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
  - synchronisation 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., \(\text{}\)-events)
- Conversational interactions can be associated to a number of interaction events:

<table>
<thead>
<tr>
<th>interaction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interaction</td>
<td>The event of initiating <em>interaction</em>.</td>
</tr>
<tr>
<td>interaction</td>
<td>The reply-event of <em>interaction</em>.</td>
</tr>
<tr>
<td>interaction</td>
<td>The commit-event of <em>interaction</em>.</td>
</tr>
<tr>
<td>interaction</td>
<td>The cancel-event of <em>interaction</em>.</td>
</tr>
<tr>
<td>interaction</td>
<td>The revoke-event of <em>interaction</em>.</td>
</tr>
</tbody>
</table>
**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 $e_1$ of type s&r and PartyR declares an interaction $e_2$ of type r&s (connected via wires to $e_1$)

- The initiation event for $e$ is:
  1. Issued by the partyS: $e \triangleleft !$
  2. Stored in a buffer of partyR,
  3. Processed by partyR and then
  4. Either executed $e \triangleleft ?$ by partyR or discarded

- The reply event for $e$ is:
  1. Issued by the partyR: $e \triangleright !$
  2. Stored in a buffer of partyS,
  3. Processed by partyS and then
  4. Either executed $e \triangleright ?$ by partyS or discarded

- The same for cancel/commit/revoke events
Events in SP: Examples

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

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

**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*)
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 of 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 $\bullet$. 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 $\neg$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 $\mathbb{I}$ to the value now (referring to when the $\boxplus$-event is sent)
Events in SP: Examples

- 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✗? ...

**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)
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

<table>
<thead>
<tr>
<th><strong>interaction</strong></th>
<th>The event of <strong>initiating interaction</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>interaction</strong></td>
<td>The <strong>reply-event</strong> of interaction</td>
</tr>
<tr>
<td><strong>interaction</strong></td>
<td>The <strong>pledge</strong> associated with interaction.</td>
</tr>
<tr>
<td><strong>interaction</strong></td>
<td>The <strong>timeout</strong> of interaction, i.e. number of units of time during which the pledge is guaranteed to hold.</td>
</tr>
<tr>
<td><strong>interaction</strong></td>
<td>The <strong>commit-event</strong> of interaction (the pledge is enforced).</td>
</tr>
<tr>
<td><strong>interaction</strong></td>
<td>The <strong>cancel-event</strong> of interaction (the pledge is discarded).</td>
</tr>
<tr>
<td><strong>interaction</strong></td>
<td>A <strong>revoke-event</strong> for interaction, which means cancelling the effects of interaction after having committed to it.</td>
</tr>
</tbody>
</table>