SRML

wires and interaction protocols

Laura Bocchi
bocchi@mcs.le.ac.uk
Agenda

- Wires
- Interaction Protocols
Often we assumed, for simplicity, that the names for the interactions and the parameters are pairwise corresponding.

In general, we can reuse specifications. This could cause mismatching of some names or even duplication (if two nodes have the same specification).
Wires and connectors

- Wires specify the correspondence between interaction/parameter names of different nodes
- E.g., SW specifies the correspondence between SP and WA
- A wire is defined as one or more connectors

<table>
<thead>
<tr>
<th>SP</th>
<th>C₄</th>
<th>SW</th>
<th>d₄</th>
<th>WA</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkShipAvail</td>
<td>S₁</td>
<td>Straight</td>
<td>R₁</td>
<td>intWR, tellShipAvail</td>
</tr>
<tr>
<td></td>
<td>I₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I₂</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>confirmShip</td>
<td>S₁</td>
<td>Straight</td>
<td>R₁</td>
<td>intCT, snd confirmShip</td>
</tr>
</tbody>
</table>

Each line represents a connector.
A connector is a triple: \( < \mu_A, P, \mu_B > \) where

- \( P \) is an interaction protocol. We use \( \text{role}_A P \) and \( \text{role}_B P \) to designate its roles and \( \text{glue}_P \) to designate the coordination.

- \( \mu_A \) and \( \mu_B \) are attachments that connect the roles of \( P \) to the signatures of the connected nodes.

(Wires and connectors)
A simple interaction protocol

INTERACTION PROTOCOL Straight is

ROLE A
s&r S₁
 allotted i₁:product
 allotted i₂:money

ROLE B
r&s R₁
 allotted o₁:product
 allotted o₂:money

COORDINATION
S₁ = R₁
S₁.i₁ = R₁.o₁
S₁.i₂ = R₁.o₂

the events associated to S₁ are the same as the events associated to R₁

a one-to-one correspondence is stated between parameters
Another interaction protocol

INTERACTION PROTOCOL RobinHood is

ROLE A
s&r S₁
 ADVISED i₁:product
RECEIVED o₁:money

ROLE B
r&s R₁
 ADVISED i₁:product
RECEIVED o₁:money

COORDINATION
S₁ = R₁
S₁.i₁=R₁.i₁
S₁.o₁ >100 ⊃ S₁.o₁=R₁.o₁-50

the data can be also elaborated

if the amount is > 100£ then the protocol RobinHood steals 50£
A simple interaction protocol

INTERACTION PROTOCOL Straight is

ROLE A
s&r $S_1$
  $\diamond$ $i_1$:product
  $\blacklozenge$ $o_1$:money

ROLE B
r&s $R_1$
  $\diamond$ $i_1$:product
  $\blacklozenge$ $o_1$:money

COORDINATION
$S_1 = R_1$
$S_1.i_1 = R_1.i_1$
$S_1.o_1 = R_1.o_1$

INTERACTION PROTOCOL Straight is

ROLE A
s&r $S_1$
  $\diamond$ $i_1$:product
  $\blacklozenge$ $i_2$:usrId
  $\blacklozenge$ $o_1$:money
  $\checkmark$ $c_1$:payData

ROLE B
r&s $R_1$
  $\diamond$ $i_1$:product
  $\blacklozenge$ $i_2$:usrId
  $\blacklozenge$ $o_1$:money
  $\checkmark$ $c_1$:payData

COORDINATION
$S_1 = R_1$
$S_1.i_1 = R_1.i_1$
$S_1.i_2 = R_1.i_2$
$S_1.o_1 = R_1.o_1$
$S_1.c_1 = R_1.c_1$

- Straight can be used only on a couple of conversational interactions (s&r and r&s) that have exactly one $\diamond$-parameter and exactly one $\blacklozenge$-parameter.

- For couples of conversational interaction with a different number of parameters we must define another interaction protocol (e.g., two $\diamond$-parameter, one $\blacklozenge$-parameter and one $\checkmark$-parameters).
A simple interaction protocol

The protocol above can be used only if the parameters are all of type product, usrId, money and payData.

If we want to use, say, destination, outdate, money and paydata we have to define another protocol.

Otherwise we can parametrize interaction protocols ...
A parametrized interaction protocol

- Straight.I(d₁)O(d₂) can be used only on a couple of conversational interactions (s&r and r&s) that have exactly one \( \& \)-parameter and exactly one \( \bullet \)-parameter...

- But it can be used also for interaction that carry other data types than product and money

INTERACTION PROTOCOL Straight.I(d₁)O(d₂) is

<table>
<thead>
<tr>
<th>ROLE A</th>
<th>ROLE B</th>
<th>COORDINATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>s&amp;r S₁</td>
<td>r&amp;s R₁</td>
<td>S₁ = R₁</td>
</tr>
<tr>
<td>i₁:d₁</td>
<td>o₁:d₁</td>
<td>S₁.i₁=R₁.o₁</td>
</tr>
<tr>
<td>i₂:d₂</td>
<td>o₂:d₂</td>
<td>S₁.i₂=R₁.o₂</td>
</tr>
</tbody>
</table>

**COORDINATION**

<table>
<thead>
<tr>
<th>Supplier</th>
<th>SP</th>
<th>c₄</th>
<th>SW</th>
<th>d₄</th>
<th>WA</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkShipAvail</td>
<td>s&amp;r</td>
<td>i₁</td>
<td>₁</td>
<td>Many</td>
<td>₁</td>
</tr>
<tr>
<td>tellShipAvail</td>
<td>r&amp;s</td>
<td>i₂</td>
<td>₂</td>
<td>Many</td>
<td>₂</td>
</tr>
<tr>
<td>confirmShip</td>
<td></td>
<td>₃</td>
<td>₃</td>
<td></td>
<td>R₁</td>
</tr>
</tbody>
</table>
Example: parameters mismatching

INTERACTION PROTOCOL Internal2SMS is

ROLE A

snd S₁

i₁:phoneNum
i₂:reference
i₃:string
i₄:geoData

ROLE B

rcv R₁

i₁:phoneNum
i₂:string

LOCAL

textify:reference,string,geoData→string

COORDINATION

S₁ = R₁
S₁.i₁ = R₁.i₁
R₁.i₂ = textify(S₁.i₂, S₁.i₃, S₁.i₄)
Example: add authentication

- What if we wish to add authentication in the interactions between WR and SP?

- Well, we could modify Supplier and Warehouse

- But what if we decide to use services for warehouses with authentication but we do not want to change Supplier?

- We can add authentication in the interaction protocol

```
INTERACTION PROTOCOL Secure is

ROLE A
s&r S₁
☑️ i₁:product
☑️ o₁:money

ROLE B
r&s R₁
☑️ i₁:product
☑️ i₂:password
☑️ o₁:money

COORDINATION
S₁ = R₁
S₁.i₁=R₁.i₁
S₁.o₁=R₁.o₁
R₁.i₂="secret"
```
Let's go back to the example...

- EX-Ps describe what the module provides
- EX-Rs describe what the module requires
- EX-Is are not a node that executes but just a description
- In fact, what CR “provides” is what SP “provides”
- In fact, WR describes what some other node in another module “provides” and we have to interact to
The protocol of the EX-P

**BUSINESS PROTOCOL** Customer is

**INTERACTIONS**

- r&s requestQuote
  - which:product
  - cost:money
- r&s orderGoods
  - many:nat
  - much:money
- rcv makePayment
- snd shipOrder

**BEHAVIOUR**

... 

**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)

**BEHAVIOUR**

... 

EX-P have the same “direction” of the node to which they are connected
The protocol of the EX-R

**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)

**BUSINESS PROTOCOL** Warehouse is

**INTERACTIONS**
- r&s checkShipAvail
  - which:product, many:nat
- snd confirmShip

**BEHAVIOUR**
...

EX-R have complementary “direction” with respect to the node to which they are connected
The wires to EX-I

<table>
<thead>
<tr>
<th>SP Supplier</th>
<th>c₁</th>
<th>SW</th>
<th>d₂</th>
<th>WA Warehouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&amp;S checkShipAvail</td>
<td>S₁</td>
<td>Straight</td>
<td>R₁</td>
<td>r&amp;S tellShipAvail</td>
</tr>
<tr>
<td>i₁, i₂</td>
<td></td>
<td></td>
<td>o₁, o₂</td>
<td>which</td>
</tr>
<tr>
<td>many</td>
<td></td>
<td></td>
<td>Ci₁</td>
<td>many</td>
</tr>
<tr>
<td>rcv confirmShip</td>
<td>S₁</td>
<td>Straight</td>
<td>R₁</td>
<td>snd confirmShip</td>
</tr>
</tbody>
</table>

Notice that SP (component) and WA (EX-R) have complementary interaction types.

The specification of the wires that connect module components to the provides-interface use a slightly different syntax.