A Business Protocol Development with Event-B

(August 2010)

J.R. Abrial[1]

This short text contains the presentation and the formal development of a small business protocol using Event-B [1]. It is made of three sections. First the requirements section where the problem requirements are informally (but precisely) defined. Then comes the second section where a refinement strategy is proposed. Finally, the formal development is presented in the third section together with checking that all requirements are taken into account.

1 Requirements

The problem consists in formally developing a business protocol between a Seller S and a Warehouse.

This protocol involves a Seller S and a Warehouse containing products	ENV-1

S would like to eventually order a product. For doing so, he asks for the reservation of certain products.

During the protocol, S may reserve products (one at a time) in the Warehouse.	FUN-1

S may also cancel the reservation of an already reserved product.

During the protocol, S may delete the reservation of products (one at a time).	FUN-2

The protocol finally terminates when S orders one of the reserved products.

S may order one of the reserved products. This ends the protocol.	FUN-3

A clerck is in charge of the Warehouse.

The protocol also involves a Warehouse Clerck C	ENV-2

Both S and C communicate by exchanging messages.

S and C communicates by means of messages.	ENV-3

For doing a reservation, S sends a specific message to C.

[1] jrabrial@neuf.fr

S makes a product reservation by sending a message to C	FUN-4

In return for a reservation message from S, C always sends a confirmation message to S.

C always confirms a reservation by sending a message to S.	FUN-5

For doing the deletion of a product reservation, S sends a specific message to C.

S deletes a reserved product by sending a message to C	FUN-6

For ordering a reserved product, S sends a specific message to C.

S orders a reserved product by sending a message to C	FUN-7

S cannot send any messages between the sending of a reservation to C and the reception of the corresponding confirmation from C.

Order, deletion, or reservation messages cannot be sent by S between a reservation message and its confirmation	FUN-8

At the end of the protocol, the ordered product must be the same for both S and C.

The order product must be the same for both partners	FUN-9

Messages sent by S can be reordered arbitrarily before being treated by C.

Messages can be reordered before being treated by C	ENV-4

However, we suppose that no messages are lost.

Messages are never lost	ENV-5

C must treat all pending messages before ending the protocol.

At the end of the protocol, all pending messages must be treated by C	ENV-6

2 Refinement Strategy

The development is made of an initial model followed by some refinements:

The initial model is a high level abstraction showing what the seller S can eventually do, namely to order a product. This covers requirement FUN-3 telling that the seller may eventually order a product.

The first refinement still corresponds to an abstraction involving S only. This model contains the definition of all actions the seller S can do: to reserve, to delete, to order, and finally to do nothing after ordering. This covers requirements FUN-1 (reservation), FUN-2 (deletion), and again FUN-3 (ordering).

The second refinement contains the introduction of the clerck C and the message exchanges between S and C. This is done by means of some channels between the two. This refinement should cover the remaining requirements.

The third refinement is a technical one implementing some conditions on the channels by means of boolean variables.

3 Formal Development

3.1 Initial Model

In this initial model, we just formalise what the seller can eventually do: order a product.
First, we define a carrier set PRD: it describes the product information. It is left completely abstract in this presentation.

sets: PRD

Then we define a single variable S_ORD denoting the set of ordered product. It is most a singleton set:

variables: S_ORD

inv0_1: $S\_ORD \subseteq PRD$

inv0_2: $S\_ORD \neq \varnothing \Rightarrow \exists x \cdot S\_ORD = \{x\}$

Finally, we define the dynamics of the system by means of two events. The INIT event making S_ORD empty at the beginning and the Order event making S_ORD a singleton set:

INIT
$S\_ORD := \varnothing$

Order
any p **where**
$p \in PRD$
$S\_ORD = \varnothing$
then
$S\_ORD := \{p\}$
end

3.2 First Refinement: Global View of the Seller

In this first refinement, we consider that both participants S and C are working in the same site, say in the Seller site. Of course, it does not correspond at all to the reality, but this gives us the possibility to simplify the formal presentation.

The State.

Besides variable S_ORD of the previous abstraction, we define two more variables: S_USD and S_RES:

- S_USD denotes the set of products that have been reserved so far by S,

- S_RES denotes the set of products that have been reserved so far and not deleted by S.

Notice that the names of these variables are all prefixed by "$S\_$": this is because the corresponding information will later be local to the S side.

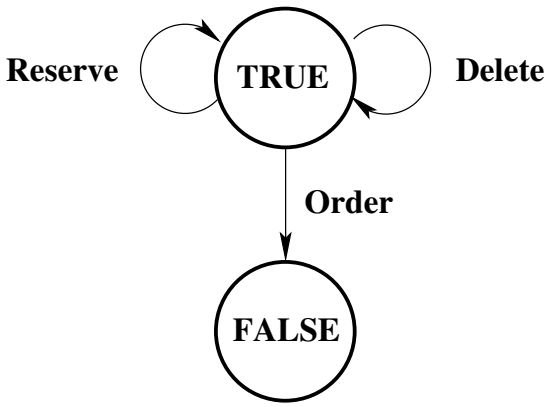
variables:	S_USD
	S_RES
	S_ORD

We now define the invariants. In **inv01_1**, the set S_USD is simply typed as a subset of PRD. We could have been more precise here by saying that S_USD is a *finite set*, but we have not done so as this finiteness property does not play any role in our formalisation. As expected, in **inv01_2**, S_RES is simply defined as a subset of S_USD (the set difference between the two corresponds to products that have been first reserved and later deleted). Finally, the set S_ORD is included in S_RES: this is expressed in invariants **inv01_3**. We already know from the initial model that S_ORD is at most a singleton set.

inv01_1:	$S\_USD \subseteq PRD$
inv01_2:	$S\_RES \subseteq S\_USD$
inv01_3:	$S\_ORD \subseteq S\_RES$

The Events.

We are now able to define our events: Reserve, Delete, and Order. In the following diagram, we show under which circumstances these events are enabled. It depends on the emptyness of variable S_ORD: TRUE corresponds to S_ORD empty (protocol active) whereas FALSE corresponds to S_ORD not empty (protocol terminated).



Notice that we have not indicated in the previous diagram that events Delete and Order also require that S_RES is not empty in order to be enabled. This is shown in the guards of these events below:

INIT
$S\_USD := \varnothing$
$S\_RES := \varnothing$
$S\_ORD := \varnothing$

Reserve
any p **where**
$p \notin S\_USD$
$S\_ORD = \varnothing$
then
$S\_USD := S\_USD \cup \{p\}$
$S\_RES := S\_RES \cup \{p\}$
end

Note that in the event Reserve the inference typing deduces from the guard $p \notin S\_USD$ that the condition $p \in PRD$ holds.

Delete
any p **where**
$p \in S\_RES$
$S\_ORD = \varnothing$
then
$S\_RES := S\_RES \setminus \{p\}$
end

Order
any p **where**
$p \in S\_RES$
$S\_ORD = \varnothing$
then
$S\_ORD := \{p\}$
end

Proofs

There are 7 invariant preservation proofs. They are all straightforward and easily proved automatically by the Rodin Platform prover [2].

Meeting the Requirements

As can be seen, the event Reserve covers FUN-1, the event Delete covers FUN2, and the event Order covers FUN_3.

3.3 Second Refinement: Introducing Messages and Channels

In this refinement, we are more realistic than in the abstract model of previous section. We are introducing the Clerck C. Now S and C have to communicate with each other. This is done by means of *messages* that are carried out from one site to the other by means of *channels*.

The State.

First, we introduce three variables C_USD. C_RES, and C_ORD. Such variables are the C counterparts of similar S variables we already encountered in the previous section:

variables: C_USD
C_RES
C_ORD

Then we introduce channels between the two sites: res_chn, cfm_chn, del_chn, and ord_chn:

- The variable res_chn denotes the reservation channel between S and C: it contains at most a single message to C with a new reserved product chosen by S.

- The variable cfm_chn denotes the confirmation channel between C and S: we shall see below that it also contains at most a single message as res_chn does.

- The variable del_chn denotes the deletion channel between S and C: it might contain several pending deletion messages sent by S but not yet treated by C.

- The variable ord_chn contains the ordering message sent by S to C. It contains the product chosen by S. This channel is artificial. Normally, such a message is also put in del_chn with an indication that it is not a deletion but an ordering. In order to avoid at this level introducing such an indication, we have defined a special channel for this. It can be later refined, thus removing ord_chn.

variables:	res_chn
	cfm_chn
	del_chn
	ord_chn

The three variables of S are not touched in this refinement. Notice that such a refinement, where abstract variables are not touched and where some new variables are added, is called an *horizontal refinement*.

Now we propose some invariants. Our first three invariants introduce variables C_USD, C_RES, and C_ORD. They are similar to invariants **inv0_1**, **inv0_2**, and **inv0_3** of previous abstraction:

inv2_1:	$C\_USD \subseteq PRD$
inv2_2:	$C\_RES \subseteq C\_USD$
inv2_3:	$C\_ORD \subseteq C\_RES$

Then we define the relationship between C_USD, S_USD and res_chn. Variable C_USD cannot be equal to S_USD because they are not in the same site: a message can be situated between the two travelling from S to C. This figure will be encountered in many occasions in similar relationship. Invariant **inv2_4** states that no information is lost: the deficit between C_USD and S_USD is compensated by what is in res_chn. Moreover, the contents of C_USD and res_chn are incompatible as indicated in **inv2_5**. This can be easily understood: what is sent from S to C is new for C. In other words, C_USD and res_chn partition S_USD. This partitioning is the basic mathematical property formalizing that some new information is travelling from one site to the other, it is fundamental.

inv2_4:	$S\_USD = C\_USD \cup res\_chn$
inv2_5:	$C\_USD \cap res\_chn = \varnothing$

We have a similar relationship between C_RES, S_RES, res_chn and del_chn. However, this case is a little more complicated than the previous one as we have two channels involved, namely res_chn and del_chn. The reserved products on the S side (in S_RES) together with the contents of the deletion channel del_chn, containing the would-be deleted products on the C side, are the same as the reserved

product on the C side (in C_RES) together with a possible product in the reservation channel res_chn, containing a would-be reserved product on the C side. Notice theorem **thm2_1** that can be deduced from invariants **inv2_5** and **inv2_2**.

inv2_6: $S\_RES \cup del\_chn = C\_RES \cup res\_chn$

inv2_7: $S\_RES \cap del\_chn = \varnothing$

thm2_1: $C\_RES \cap res\_chn = \varnothing$

Our last partitioning relationship deals now with C_ORD, S_ORD, and ord_chn:

inv2_8: $S\_ORD = C\_ORD \cup ord\_chn$

inv2_9: $C\_ORD \cap ord\_chn = \varnothing$

Finally, we need to type the confirmation channel cfm_chn:

inv2_10: $cfm\_chn \subseteq PRD$

Clearly, more invariants could have been defined but they are not useful for our main purpose, i.e. guaranteeing that the requirements are fulfilled.

The Events

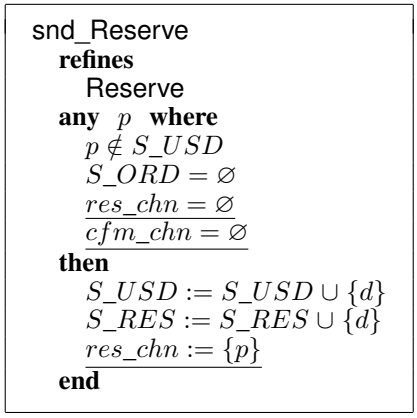
We are now ready to define our events. They are the following on the S side:

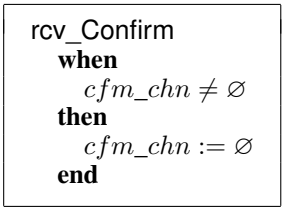
- snd_Reserve refining the abstract event Reserve
- rcv_Confirm (new event)
- snd_Delete refining the abstract event Delete
- snd_Order refining the abstract event Order

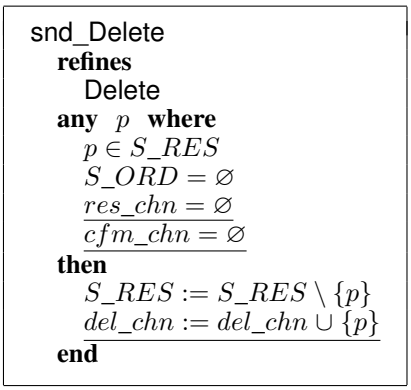
and the following new events on the C side:

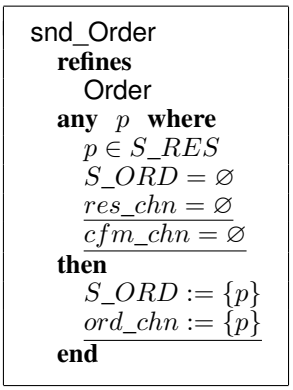
- rcv_Reserve
- rcv_Delete
- rcv_Order

Note that on the C side we have no explicit event sending a confirmation after receiving a reservation: it is done directly by the event rcv_Reserve. In the refined versions below (events snd_Reserve, snd_Delete, and snd_Order), the underlined statements are the new ones that are added: more guards and more actions (this constitutes the essence of horizontal refinements).

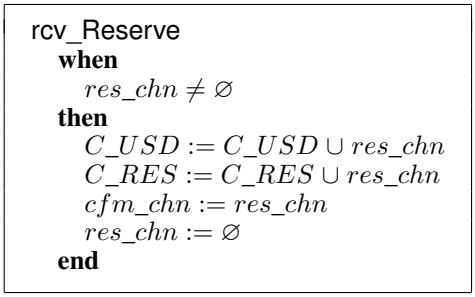


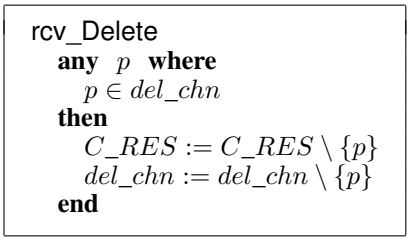


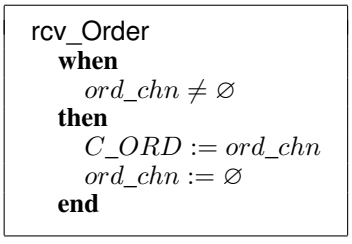




Here are finally the C events:







Proofs

There are 29 invariant preservation proofs in this refinement.They are all straightforward and easily proved automatically by the Rodin Platform prover [2] except three of them.

First, we cannot prove that the event rcv_Delete preserves invariant **inv2_6**, namely:

inv2_6: $S\_RES \cup del\_chn = C\_RES \cup res\_chn$

This amounts to proving the following:

$$\begin{array}{l} p \in del\_chn \\ \Rightarrow \\ S\_RES \cup (del\_chn \setminus \{p\}) = (C\_RES \setminus \{p\}) \cup res\_chn \end{array}$$

According to **inv2_7** (i.e. $S\_RES \cap del\_chn = \varnothing$), it is easy to prove:

$$S\_RES \cup (del\_chn \setminus \{p\}) = (S\_RES \cup del\_chn) \setminus \{p\}$$

Hence, according to **inv2_6**, we are left to prove:

$$(C\_RES \cup res\_ch) \setminus \{p\} = (C\_RES \setminus \{p\}) \cup res\_chn$$

But here, we cannot conclude. Since p belongs to del_chn, thus, according to **inv2_6**, it belongs either to C_RES or to res_chn (but not to both, remember **thm2_1**: $C\_RES \cap res\_chn = \varnothing$) . If p belongs to C_RES, we are done, but if p belongs to res_chn we cannot conclude. In order to be sure that p does not belong to res_chn, it is sufficient to introduce the following invariant, which is very intuitive: a newly reserved product (in res_chn) cannot be yet deleted (in del_chn)

inv2_11: $res\_chn \cap del\_chn = \varnothing$

This new invariant makes the previous proof automatically discharged and the preservation of this new invariant is also discharged automatically.

Then we cannot prove that the event rcv_Order preserves invariant **inv2_3**, namely:

inv2_3: $C\_ORD \subseteq C\_RES$

This amounts to proving:

$$ord\_chn \subseteq C\_RES$$

We, simply add this statement as a new invariant:

inv2_12: $ord\_chn \subseteq C\_RES$

As before, this new invariant makes the previous proof automatically discharged and the preservation of this new invariant is also discharged automatically.

Finally, we cannot prove that the event rcv_Order preserves invariant **inv2_8**, namely:

inv2_8: $S\_ORD = C\_ORD \cup ord\_chn$

This amounts to proving the following:

$$ord\_chn \neq \varnothing \ \Rightarrow \ S\_ORD = ord\_chn$$

Again, we simply add the following invariants:

inv2_13: $ord\_chn \neq \varnothing \ \Rightarrow \ S\_ORD = ord\_chn$

After this, all proofs are automatically discharged by the Rodin Platform prover. We now have 42 proofs.

Meeting the Requirements

Let us now see how our remaining requirements have been achieved:

- Requirement FUN-4 (reservation) is achieved by the event snd_Reserve which sends a reservation message.

- Requirement FUN-5 (a request has to be confirmed) is achieved by the event rcv_Reserve which sends a confirmation.

- Requirements FUN-6 and FUN-7 (deletion and ordering)) is achieved by events snd-Delete, rcv-Delete, snd-Order, and rcv-Order.

- Requirement FUN-8 (nothing can be done by S while waiting for a confirmation) is achieved by the presence of the guards $res\_chn = \varnothing$ and $cfm\_chn = \varnothing$ in the snd_ events on the S side.

- Requirement FUN-9 (the ordered product must be the same for both partners) cannot be shown to be achieved unless one adds the following, which is a theorem:

thm2_2: $ord\_chn = \varnothing \ \Rightarrow \ S\_ORD = C\_ORD$

This theorem is proved automatically. This makes all together 43 proofs for this refinement.

- Requirement ENV-1 and ENV-2 (the protocol involves two partners S and C) is achieved by introducing S in the initial model and C in the refinement.

- Requirement ENV-3 (messages) is achieved by the various snd_ and rcv_ events.

- Requirement ENV-4 (pending deletions or order can be reordered) is achieved by the fact that del_chn is a set (no ordering) and that C can either chose to receive a deletion or an order when there are pending messages in both channels.

- Requirement ENV-5 (no loss of messages) is achieved by the fact that the only way to remove a message from a channel is by receiving it.

- Requirement ENV-6 (pending messages can always be treated) is achieved by the fact that the receiving events (those whose names are prefixed with "rcv_") have no additional guards besides the one concerned with the relevant channels.

3.4 Third Refinement: Towards an Implementation

The State Another refinement can be added to replace the guards - those involving the emptyness of some channels in the snd events on the S side - by corresponding boolean variables. For this, we introduce two boolean variables on the S side:

variables: S_wrk
S_res

together with the following invariants:

inv3_1: $S\_wrk = \mathrm{bool}\,(S\_ORD = \varnothing)$

inv3_2: $S\_res = \mathrm{bool}\,(res\_chn = \varnothing \ \wedge \ cfm\_chn = \varnothing)$

The Events The following events only are modified in this refinement, as indicated below:

snd_Reserve
refines
Reserve
any p **where**
$p \notin S\_USD$
$\underline{S\_wrk = \mathrm{TRUE}}$
$\underline{S\_res = \mathrm{TRUE}}$
then
$S\_USD := S\_USD \cup \{d\}$
$S\_RES := S\_RES \cup \{d\}$
$res\_chn := \{p\}$
$\underline{S\_res := \mathrm{FALSE}}$
end

rcv_Confirm
when
$cfm\_chn \neq \varnothing$
then
$cfm\_chn := \varnothing$
$\underline{S\_res := \mathrm{TRUE}}$
end

snd_Delete
refines
Delete
any p **where**
$p \in S\_RES$
$\underline{S\_wrk = \mathrm{TRUE}}$
$\underline{S\_res = \mathrm{TRUE}}$
then
$S\_RES := S\_RES \setminus \{p\}$
$del\_chn := del\_chn \cup \{p\}$
end

snd_Order
refines
Order
any p **where**
$p \in S\_RES$
$\underline{S\_wrk = \mathrm{TRUE}}$
$\underline{S\_res = \mathrm{TRUE}}$
then
$S\_ORD := \{p\}$
$ord\_chn := \{p\}$
$\underline{S\_wrk := \mathrm{FALSE}}$
end

The Proofs This refinement requires 15 proofs, all proved automatically except one. The event rcv_Confirm cannot preserve invariant **inv3_2**:

inv3_2: $S\_res = \mathrm{bool}\,(res\_chn = \varnothing \ \wedge \ cfm\_chn = \varnothing)$

It amounts to proving:

$$cfm\_chn \neq \varnothing \ \Rightarrow \ res\_chn = \varnothing$$

We just introduce this statement as a new invariant and that solves the problem:

inv3_3: $cfm\_chn \neq \varnothing \ \Rightarrow \ res\_chn = \varnothing$

We now have 19 proofs, all discharged automatically.

3.5 Decomposition

After this refinement, it can be observed that the S events are dealing with the S variables only and the four channels res_chn (writing), cfm_chn (reading), del_chn (writing), ord_chn (writing). Similarly, the C events are dealing with the C variables only and the four channels res_chn (reading), cfm_chn (writing), del_chn (reading), ord_chn (reading). As a consequence, it is possible now to *decompose* this system. By introducing intermediate buffers, the decomposition can be made as follows: the S component, the C component, the channel component (middleware).

4 Conclusion

We presented a short development in Event-B, which has been ported to the Rodin Platform [2]

References

1. J.R. Abrial. *Modeling in Event-B: System and Software Engineering* Cambridge University Press (2010)
2. http: //www.event-b.org *Rodin Platform*