The Invoice Order system case study

in AsmetaL

Riferimento

Egon Boerger and Angelo Gargantini and Elvinia Riccobene
ASM
in Software Specification Methods, An Overview Using a Case Study Edited by: Henri Habrias and France Marc Frappier,

ISBN: 1905209347 (2006)

Disponibile dalla mia home page

Invoicing orders (M. Allemand et al. (eds.). Comparing Techniques, ISBN 2-906082-29-5. IRIN Nantes, March 1998) : The Problem

- R1: The subject is to invoice orders.
- R2: To invoice is to change the state of an order (to change it from the state "pending" to "invoiced").
- R3: On an order, we have one and one only reference to an ordered product of a certain quantity. The quantity can be different to other orders.
- R4: The same reference can be on several different orders.
- R5: The state of the order will be changed into "invoiced" if the ordered quantity is <= to the quantity which is in stock according to the reference of the ordered product.

Invoicing orders (M. Allemand et al. (eds.). Comparing Techniques, ISBN 2-906082-29-5. IRIN Nantes, March 1998) : The Problem

R6: All the ordered references are references in stock. **The stock or the set of the orders may vary due to:**

- **the entry of new orders or cancelled orders**
- **having a new entry of quantities of products in stock** at the warehouse

You have to consider the two following cases:

(a) Case 1

- You **do not have to take these entries into account**. This means that you will not receive two entry flows (orders, entries in stock). **The stock and the set of orders are always given to you in a up-to-date state**.

(b) Case 2

- You have to **take into account the entries of:**
  - **new orders**
  - **cancellations of orders**
  - **entries of quantities in the stock**

ASM signature -Case 1

Domains:

- **abstract domain** Orders set of orders - by R1, static by R6a
- **domain** Quantity **subsetof** Natural the quantity values, by R3
- **abstract domain** Products set of orders - by R3
- **enum domain** Order_status= {INVOICED|PENDING}

Functions:

- **dynamic monitored** referencedProduct: Orders -> Products
  the product referenced in an order - by R3
- **dynamic monitored** orderQuantity: Orders ->Quantity
  returns the quantity in the order - by R3
  not injective, not constant – by R4
- **dynamic controlled** stockQuantity: Products -> Quantity
  the quantity of products in stock - by R5
  Assumption: the stock is only updated by the system
- **dynamic controlled** orderState: Orders -> Order_status
  the status of an order – dynamic controlled by R2 and R5

ASM transition rules

To invoice an order at a time:

By R2,R5 there is only one transition to change the state of an order

A **single-order rule** can be as follows: per step at most one order is invoiced, with an unspecified schedule (**not taking into account any arrival time of orders**) and with a deletion function under the assumption that stockQuantity is updated only by invoicing.

```
rule r_InvoiceSingleOrder =
  choose $order in Orders with orderState($order) = PENDING and
          orderQuantity($order) <=
          stockQuantity(referencedProduct($order))
  do par
     orderState($order) := INVOICED
     r_DeleteStock[referencedProduct($order),orderQuantity($order)]
  endpar


rule r_DeleteStock($p in Products ,$q in Quantity)=
```

ASM transition rules

Other strategies to simultaneously invoice a certain number of orders for one product at a time:

In case all orders for one product are simultaneously invoiced (or none if the stock cannot satisfy the request), a **all-or-none** strategy can be expressed by the following rule **InvoiceAllOrNone**:

Function **pendingOrders** yields the set of pending orders for a certain product, while the (static) function **totalQuantity** returns the total quantity of a set of orders.

```
rule r_InvoiceAllOrNone =
  choose $product in Products do
    let ( $pending = pendingOrders($product) ) in
      let ( $total = totalQuantity($pending) ) in
        if $total <= stockQuantity($product) then
          par
            forall $order in $pending do
                    orderState($order) := INVOICED
                     r_DeleteStock[$product, $total]
          endpar
```

First strategy

Auxiliary functions

```
static function
pendingOrders($p in Products): Powerset(Orders) =
{$o | $o in Orders with orderState($o) = PENDING and
referencedProduct($o) = $p}

static function
totalQuantity($so in Powerset(Orders)): Quantity =
if (isEmpty($so)) then 0
else let $first = first(asSequence($so) in
quantity($first) + totalQuantity(excluding($so,$first))
endif
```

Insieme finito di ordini

ASM transition rules

Other strategies to simultaneously invoice a certain number of orders for one product at a time:

To avoid the *deadlock* when the stock cannot satisfy any request, the following rule **InvoiceOrdersForOneProduct** introduces some non determinism in the choice of a set of pending orders which can be invoiced according to the available quantity in stock.

```
rule r_InvoiceOrdersForOneProduct =
  choose $product in Products do
    let ($pending = pendingOrders($product)) in
      choose $orderSet in Powerset($pending) with
        totalQuantity($orderSet) <= stockQuantity($product) do
          par                                                    Second strategy
           forall $order in $orderSet
                  do orderState($order) := INVOICED
                   r_DeleteStock[$product, totalQuantity($orderSet)]
          endpar
     endlet
```

ASM transition rules

Other strategies to simultaneously invoice a certain number of orders **for all products** at a time:

To parallelize invoicing orders over all products, a slight variant of the previous rule can be obtained replacing the **choose** $product **in** Products with **forall** $product in Products.

Third strategy

To further maximize a product quantity invoiced at the time, a new strategy, the rule **InvoiceMaxOrdersForOneProduct,**consists in choosing a maximal invoicable subset of simultaneously invoiced pending orders for the same product.

For this rule we need to define a static function **maxQuantitySubsets** :
Powerset(Powerset(Orders)) ->
Powerset(Powerset(Orders))
which, given a set of set of orders, returns the set of all the sets which have a maximum quantity.

Fourth strategy

ASM transition rules

Other strategies to simultaneously invoice a certain number of orders at a time:

```
rule r_InvoiceMaxOrdersForOneProduct =
  choose $product in Products do
    let ($pending = pendingOrders($product)) in
      let ($invoicablePending = {$o in Powerset($pending) |
            totalQuantity($o) <= stockQuantity($product) : $o} ) in
        choose $orderSet in maxQuantitySubsets($invoicablePending) do
          par
            forall $order in $orderSet do
                  orderState($order) := INVOICED
                   r_DeleteStock[$product, totalQuantity($orderSet)]
          endpar
        endlet
    endlet
```

Fourth strategy

ASM transition rules

Another strategy not driven by a first choice of a product: choose a set of pending orders, with enough referenced products in the stock, to be simultaneously invoiced.

The predicate **invoicable** is true on a set of pending orders with enough quantity of requested products in the stock, and a function **refProducts** (recursively defined) yields the set of all products referenced in a set of orders.

```
rule r_InvoiceOrders =
 choose $orderSet in Powerset(Orders) with invoicable($orderSet) do
    par
      forall $order in $orderSet do
            orderState($order) := INVOICED
      forall $product in referencedProducts($orderSet) do
            r_DeleteStock[$product, totalQuantity($orderSet,$product)]
    endpar
```

Last strategy

Auxiliary functions

```
static function
invoicable($so in Powerset(Orders)) : Boolean =
forall $o in $so with orderState($o) = PENDING and
forall $p in Products with totalQuantity($so,$p) <=
                                        stockQuantity($p)

static function
refProducts($so in Powerset(Orders)) : Powerset(Products) =
if (isEmpty($so)) then {}
else let $first = first(asSequence($so) in
        including(refProducts(excluding($so,$first)),
                                referencedProduct($first))
endif
```

ASM main rule and initial state

```
/*------- main rule   --------*/

main rule r_ordersystem =
        r_InvoiceSingleOrder[]
```

One can assume that all the orders are initially pending:

```
default init s_1:
        function orderState($o in Orders) = PENDING
```

ASM signature - Case 2

Domains (changed):

- **dynamic abstract domain** Orders set of orders - by R1
  dynamic by R6b
- **enum domain** Order_status= {INVOICED|PENDING|CANCELLED}
  We assume: cancelled orders are not deleted, but their status changed to CANCELLED

The domains Orders and Products and all the functions for case 1 remain.

New Functions:

- **dynamic monitored** newOrders: Seq(Prod(Products,Quantity))
  the sequence of orders to add (a sequence of pairs product and quantity)
- **dynamic monitored** ordersToCancel: Seq(Orders) the sequence of orders to cancel, and
- **dynamic monitored** newItems: Seq(Prod(Products,Quantity))
  the new quantities to add in the stock (sequence of pairs product and quantity)

ASM rules - Case 2

New rules:

Besides the action of invoicing an order, R6b introduces other three operations:

(1) cancellation of orders **r_CancelOrders**,

(2) insertion of new orders **r_AddOrders**,

(3) addition of quantities of products in the stock **r_AddItems**.

We assume that these operations are driven by the three new monitored functions.

ASM rules - Case 2

New rule:

```
/*--- cancellation of orders ---*/
rule r_CancelOrders =
    forall $order in asSet(ordersToCancel) do
      orderState($order) := CANCELLED
```

ASM rules - Case 2

New rule:

```
/* --- incoming orders --- */
rule r_AddOrders =
  forall $pair in asSet(newOrders) do
   let ($product = first($pair),
        $quantity = second($pair)) in
      extend Orders with $order do
         par
            referencedProduct($order):= $product
            orderQuantity($order):= $quantity
            orderState($order):= PENDING
         endpar
   endlet
```

ASM rules - Case 2

New rule:

```
/*--- inserting new items in stock ---*/
rule r_AddItems =
    forall $item in asSet(newItems) do
     let ($product = first($item),
          $quantity = second($item)) in
            stockQuantity($product) :=
            stockQuantity($product) + $quantity
    endlet
```

ASM main rule - Case 2

```
/*------- main rule   --------*/

main rule r_ordersystem2 =
    seq
      par
        r_AddOrders[]
        r_CancelOrders[]
        r_AddItems[]
      endpar
      r_InvoiceSingleOrder[]
    endseq
```

N.B. **r_InvoiceSingleOrder** updates the functions orderState and stockQuantity, hence it **cannot be executed in parallel with rules CancelOrders and AddItems**.