This separate appendix for the FACS 2010 article is available at the Kmelia website[5].

A The Kmelia Specification of the Stock System

The example is a simplified *Stock Management* application including a vending main service. This process manages product references (catalog) and product storage (stock). Administrators have specific rights, they can add or remove references under some consistency business rules such as: *a new reference must not be in the catalog* or *a removable reference must have an empty stock level*.

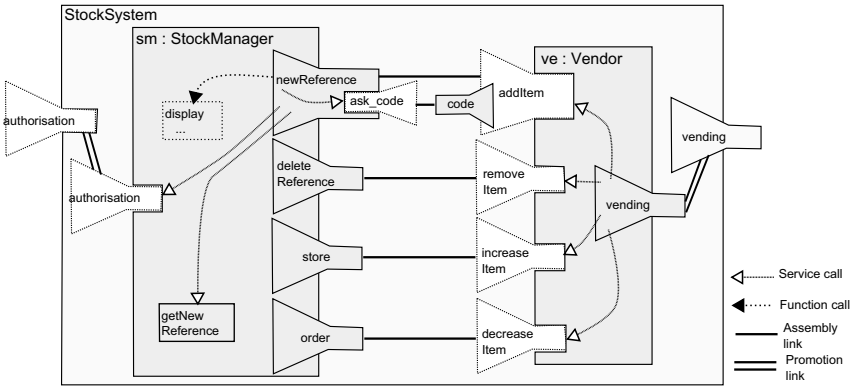

Fig. 3. Simplified Assembly of the Stock Case Study

The system is designed as a reusable component type StockSystem. It encapsulates an assembly of two components: sm:StockManager and ve:Vendor. The former is the core business component to manage references and storage. The latter is the system access interface. The main vending service is promoted at the StockSystem level. In this paper we focus only on the addItem and newReference services. According to the vending service, a user may add a new item in the stock management using the required service addItem of the Vendor component.

A.1 The StockSystem Composite Specification

Listing 4. Kmelia specification StockSystem

```
COMPONENT StockSystem
INTERFACE
   provides : {vending}
   requires : {authorisation}
SERVICES
provided vending () From ve.vending
End
required authorisation() From sm.authorisation
End
```

[5] http://www.lina.sciences.univ-nantes.fr/coloss/download/facs10_app.pdf

```
END_SERVICES
COMPOSITION
  Assembly
    Components
      sm : StockManager;
      ve : Vendor
    Links //////////assembly links/////////
      @lref: p–r sm.newReference ve.addItem
          context mapping
            ve.catalogEmpty = isEmpty(sm.catalog),
            ve.catalogFull = (size(sm.catalog) = maxInt)
          sublinks : {lcode}
      @lcode: r–p sm.ask_code ve.code
      //...
  End // assembly
END_COMPOSITION
```

A.2 The Vendor Component Partial Specification

Listing 5. Kmelia specification Vendor

```
COMPONENT Vendor
INTERFACE
  provides : {vending}
  requires : {addItem, removeItem, increaseItem, decreaseItem}
USES {STOCKLIB}
VARIABLES
  obs orders : setOf ProductItem;  # observable user card
  vendorId : Integer               # vendor personal code
INITIALIZATION
  orders := emptySet;
  vendorId := noID
SERVICES
########### provided services
# The main (provided) service is vending.

provided vending ()
Interface
   subprovides : {code}
   extrequires : {addItem, removeItem, increaseItem, decreaseItem}
Pre true
Variables # local to the service
  choice : CommandChoice ;     # command choice : addItem, ...
  ref : Integer;               # product reference given by the user
  qty : Integer;               # product quantity given by the user
  desc : String;               # product description given by the user
  pi : Integer;
  res : Boolean;
Behavior // The behaviour is specified as an infinite loop
Init i # i is the initial state
Final f # f is a final state
{ i —— {
       displayMenu();                    # call an internal action
       display("Please enter your choice");
       choice := readCommandChoice()     # call an internal action
       } —> e0,
  e0 —[choice = stop] display("bye bye") —> f,
  //final state = end of vending
  e2 —[choice = add] _addItem!!addItem() —> e10,
  e0 —[choice <> stop] display("Product reference") —> e1,
  e1 —— ref:=readInt() —> e2,
  e2 —[choice = remove] _removeItem!!removeItem(ref) —> e20,
  e2 —[choice = store] {_increaseItem!!increaseItem(ref,readInt())} —> e30,
```

```
  e2 —[choice = order] _decreaseItem!!decreaseItem(ref,readInt()) —> e40,
  //—— add Item
  e10 <<code>>, #subservice code is available here
  e10 — {
        desc=readString(); // product description
        _addItem ! msg(desc)
        } —> e11,
  e11 — _addItem??addItem(pi) —> e12,
  e12 — { if (pi <> noReference)
          then display("New reference : "+IntegerAsString(pi))
          endif } —> i,
  //—— other choices
  e20 <<code>>, #subservice code is available here
  #e20 — display("removeItem Not implemented") —> i,
  #do not test the contract
  e20 — _removeItem??removeItem(res) —> e21,
  e21 — [res] display("Done") —> i,
  e21 — [not res] display("Refused") —> i,
  #e30 — display("increaseItem Not implemented") —> i,
  #do not test the contract
  e30 — _increaseItem??increaseItem(pi) —> i,
  #e40 — display("decreaseItem Not implemented") —> i
  #do not test the contract
  e40 — _decreaseItem??decreaseItem(pi) —> i
}
Post
  obs true
End

provided code() : Integer
/* daemon service that answers the code of the user */
  Pre obs true
Behavior
Init e0
Final f
{ e0 — [vendorId = noID] display("Enter your vendor code") —> e1,
  e0 — [vendorId <> noID] __CALLER!!code(vendorId) —> f,
  e1 — vendorId=readInt() —> e0
}
Post obs Result <> noID
End
########### required services (partial description)
required addItem () : Integer
Interface
  subprovides : {code}
  Virtual Variables
     catalogFull : Boolean;
     catalogEmpty : Boolean     //possibly catalogSize
  Virtual Invariant not(catalogEmpty and catalogFull)
  Pre not catalogFull
  //No LTS
  Post (Result <> noReference) implies (not catalogEmpty)
End
required removeItem(pid : Integer) : Boolean
End
required increaseItem (pid : Integer; pqty : Integer) : Integer
End
required decreaseItem (pid : Integer; pqty : Integer) : Integer
End
END_SERVICES
```

A.3 The IntDictionary Component Specification

Listing 6. Kmelia specification IntDictionary

```
COMPONENT IntDictionary
INTERFACE
  provides : {newEntry, deleteEntry, getEntry, setEntry}
  //requires : {}
USES {STOCKLIB}
VARIABLES
  obs keys : setOf Reference;
  obs values : array [Reference] of Integer
INVARIANT
  obs @borned: size(keys)<=maxRef,
  @referenced: forall ref : Reference | includes(keys,ref) implies
     (values[ref]>=0),
  @notreferenced: forall ref : Reference | excludes(keys,ref) implies
     (values[ref] = noValue)
INITIALIZATION
  keys := emptySet;
  values := arrayInit(values,noValue);  //..empty keys
SERVICES
########## provided services
provided newEntry () : Integer  //Result = a key or noValue
Interface
   intrequires : {getNewEntry}
Pre
   size(keys) < maxRef #the keys is not full
Variables # local to the service
  res:Reference;
Initialization
  res := noValue;
Behavior
Init i # the initial state
Final f # a final state
{  i -- res := __SELF!!getNewEntry() --> e1,
  e1 -- {keys := including(keys,res); //add new reference
         values[res] := 0 //default stock is null
        }--> end,
  end -- __CALLER!!newEntry(res) --> f
       # the caller is informed from the Result and the service ends.
}
Post
  @resultRange: ((Result>=1 and Result<=maxRef) or (Result = noValue)),
  @resultValue: (Result <> noValue) implies (notIn(old(keys),Result)
      and keys = add(old(keys),Result)),
  @noresultValue: (Result = noValue) implies Unchanged{keys},
  local @refAndValue: (Result <> noValue) implies
         (values[Result] = 0 and
         (forall i : Reference | (i <> Result) implies
            (values[i] = old(values)[i])) ),
  @NorefAAndValue: (Result = noReference) implies Unchanged{values}
End

provided deleteEntry(key : Integer) : Boolean
Pre
  obs includes(keys,key)
Behavior
Init i
Final f
{  i -- { values[key] := noValue ;
          keys := excluding(keys,key)} --> res,
  res -- __CALLER!!deleteEntry(true) --> f
}
Post
  @keys: keys = excluding(old(keys),key),
  local @refRemoved:  values[key] = noValue,
  local @refUnchanged:  (forall i : Reference | (i <> key) implies
            (values[i] = old(values)[i]))
End
query provided getEntry (key : Integer) : Integer
Pre
```

```
  obs includes(keys,key)
Behavior
Init i
Final f
{  i -- __CALLER!!getEntry(values[key]) --> f
}
Post
  obs @res: Result = values[key]
End

provided setEntry (key : Integer; value : Integer) : Integer
Pre
  obs @rng: (1<=key and key<=maxRef),
  obs @val: value>=0,
  obs @add not(includes(keys,key)) implies size(keys) < maxRef
Behavior
Init i
Final f
{  i -- values[key] := value --> res,
  res -- __CALLER!!setEntry(key) --> f
}
Post
  obs @res: Result = key,
  obs @keys: keys = including(old(keys),key),
  obs @refUpdated:  values[key] = value,
  local @refUnchanged:  (forall i : Reference | (i <> key) implies
            (values[i] = old(values)[i]))
End

provided addToEntry (key : Integer; value : Integer) : Integer
Pre
  obs @key: includes(keys,key),
  obs @val: values[key]+value>=0
Behavior
Init i
Final f
{  i -- values[key] := values[key] + value --> res,
  res -- __CALLER!!addToEntry(key) --> f
}
Post
  obs @res: Result = key,
  obs @keys:  Unchanged{keys},
  obs @refUpdated:  values[key] = old(values)[key] + value,
  local @refUnchanged:  (forall i : Reference | (i <> key) implies
            (values[i] = old(values)[i]))
End

//////////internal provided services///////////////
query provided getNewEntry () : Reference
Pre
  size(keys) < maxRef #possible if the keys is not full
Variables
  i: Reference;
Initialization
  i := 1 ;
Behavior
Init i
Final f
{  i -- {while (values[i] <> noReference) do
          i := i +1
        endwhile} --> res,
  res -- __CALLER!!getNewEntry(i) --> f
}
Post
  notIn(keys,Result) //the reference is really new
End
########### required services (partial description)
END_SERVICES
```

A.4 The Manager Component Specification

Listing 7. Kmelia specification Manager

```
COMPONENT Manager
INTERFACE
  provides : {newReference, deleteReference, getInfo, order}
  requires : {authorisation, newRefCat, deleteRefCat, getRefCat,
                deleteRefStock, getRefStock, setRefStock}
USES {STOCKLIB}
VARIABLES
  vendorCodes : setOf Integer; //authorised administrators
  obs keys : setOf Reference; // product id = index of the arrays
INVARIANT
  obs @borned: size(keys) <=maxRef
INITIALIZATION
  keys := emptySet;
  vendorCodes := emptySet;      //filled by a required service
SERVICES
########## provided services
provided newReference () : Integer  //Result = ProductId or noReference
Interface
   calrequires : {ask_code} #required from the caller
   extrequires : {newRefCat, setRefCat, setRefStock}
Pre
   size(keys) < maxRef #the keys is not full
Variables # local to the service
  c : Integer;     # c : input code given by the user
  ref,key:Reference;
  d : String;          # product description
Initialization
  ref := noValue;
Behavior
Init i # the initial state
Final f # a final state
{   i -- c := __CALLER!!ask_code() --> e1,
       # gets the password on the ask_code (service) channel
  e1 -- [not(c in vendorCodes)]
        display("adding a reference is not allowed") --> end,
  e1 -- [c in vendorCodes] __CALLER ? msg(d) --> e2,
       # gets the product description
  e2 -- [d = emptyString]
        display("adding an EmptySet description is not allowed") --> end,
  //set the product description with the one given by the CALLER
  e2 -- [d <> emptyString] ref := _newRefCat!!newRefCat() --> e3,
  e3 -- key := _setRefCat!!setRefCat(ref,d) --> e4,
  //default stock is null
  e4 -- [key = ref] key := _setRefStock!!setRefStock(key,0) --> e5,
  e4 -- [key <> ref] display("reference error") --> end,
       # consistency to check
  e5 -- keys := including(keys,key) --> end,   //add new reference
 end -- __CALLER!!newReference(key) --> f
       # the caller is informed from the Result and the service ends.
}
Post
  @resultRange: ((Result >=1 and Result <=maxRef) or (Result = noReference)),
  @resultValue: (Result <> noReference) implies (notIn(old(keys),Result) and
       keys = add(old(keys),Result)),
  @noresultValue: (Result = noReference) implies Unchanged{keys}
End

provided deleteReference(pid : Integer) : Boolean //...
Interface
   calrequires : {ask_code} #required from the caller
   extrequires : {deleteRefCat, deleteRefStock}
Variables # local to the service
  c,qty : Integer;     # c : input code given by the user
```

```
  resCat, resSto:Boolean;
Pre
  obs @keys: includes(keys,pid)
Behavior
Init i
Final f
{  i -- c := __CALLER!!ask_code() --> e1,
       # gets the password on the ask_code (service) channel
  e1 -- [not(c in vendorCodes)]
        display("adding a reference is not allowed") --> end,
  e1 -- [c in vendorCodes] nop --> e2,
  e2 -- resCat := _deleteRefCat!!deleteRefCat(pid) --> e3,
  e3 -- resSto := _deleteRefStock!!deleteRefStock(pid) --> end,
     // reverse bas delete ??
  end -- __CALLER!!deleteReference(resCat and resSto) --> f
}
Post
  Unchanged{keys}
End

query provided getInfo(pid : Integer) : ProductItem //...
Interface
   extrequires : {getRefCat, getRefStock}
Pre
  obs @keys: includes(keys,pid)
Variables # local to the service
  pi : ProductItem;      # c : input code given by the user
  resCat, resSto:Boolean;
Behavior
Init i
Final f
{  i -- pi.id := pid --> e1,
   e1 -- pi.desc := _getRefCat!!getRefCat(pid) --> e2,
   e2 -- pi.quantity := _getRefStock!!getRefStock(pid) --> end,
   end -- __CALLER!!getInfo(pi) --> f
}
Post
  Unchanged{keys}
End

provided order (pid : Integer; pqty : Integer) : Integer //
Interface
   extrequires : {addRefStock}
Variables # local to the service
  qty : Integer;      # c : input code given by the user
  res:Reference;
Pre
  obs @keys: includes(keys,pid),
  obs @qty: pqty <> 0
Behavior
Init i
Final f
{  i -- [pqty > 0] qty := - pqty --> e1,
   i -- [pqty < 0] nop --> e1,
  e1 -- res := _addRefStock!!addRefStock(pid,qty) --> end,
  end -- __CALLER!!order(res) --> f
}
Post
  Unchanged{keys}
End
########### required services (partial description)
required ask_code() : Integer
  //default assertion = true and No LTS
End
required newRefCat() : Integer //...
End
required deleteRefCat(pid : Integer) : Boolean //...
End
```

```
provided getRefCat (pid : Integer) : String //...
End
required setRefCat(pid : Integer; pqty : String) : Integer //...
End
required authorisation() : setOf String  //...
End
required deleteRefStock(pid : Integer) : Boolean //...
End
required getRefStock(pid : Integer) : Integer //...
End
required setRefStock(pid : Integer; pqty : Integer) : Integer //...
End
required addRefStock(pid : Integer; pqty : Integer) : Integer //...
End
END_SERVICES
```

A.5 Another design of the StockManager Composite

From a methodology point of view, the service of a composite should never directly call services of sub components, they should rather call promotions of those services, which need not to be put in the component interface if it is not useful because the promotion concept is orthogonal to the observability.

Here is an example of design of the StockManager Composite in the spirit of Kmelia.

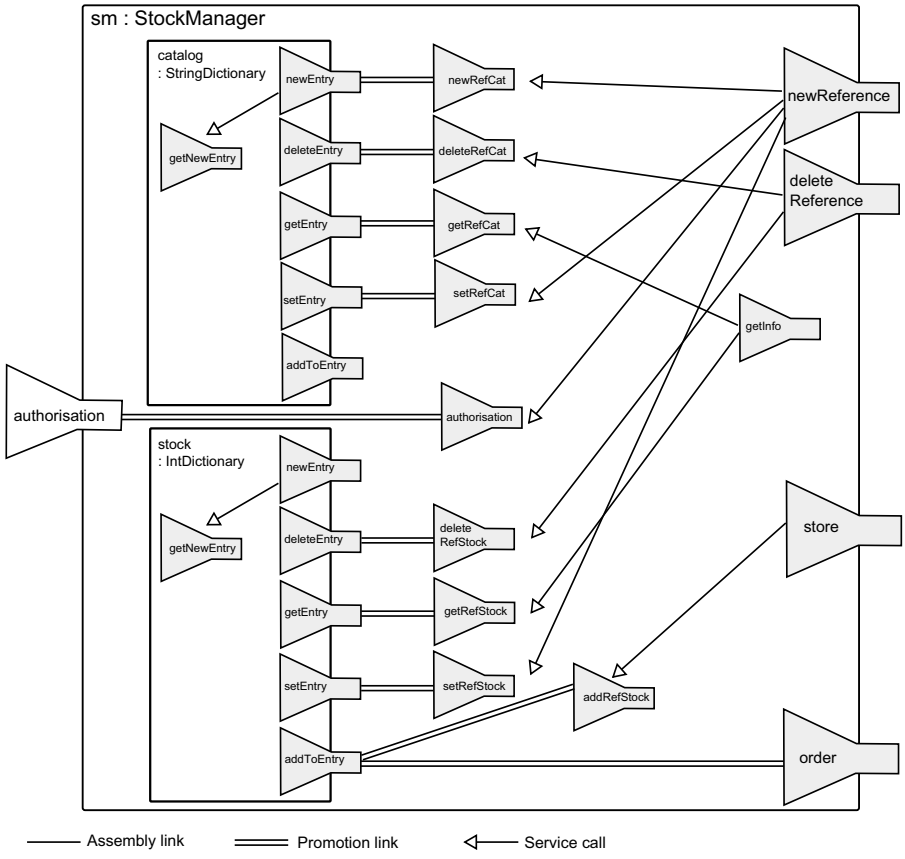

Fig. 4. Stock Manager composite component without controller sub-component

B The derived B models

B.1 The B machine StockLib

```
/* StockLib
 × Author∈
 × Creation date∈ mar. mai 5 2009
 */
MACHINE
  StockLib
SEES
  Default
CONSTANTS
  MaxRef,
  References,
  NoQuantity,
  NoReference
PROPERTIES
  MaxRef = 100 ∧
  References = 1..MaxRef ∧
```

```
  NoQuantity = −2 ∧
  NoReference = −3
END
```

B.2 IntDictionary

```
/* IntDictionary
 × Author∈ Messabihi/Andre
 × Creation date∈ ven. juil . 23 2010
 */
MACHINE
    IntDictionary

SETS
    String ;
    CommandChoice = { add , remove , store , order , stop }

CONCRETE_CONSTANTS
    noValue ,
    maxRef ,
    noReference ,
    noQuantity ,
    noID ,
    maxInt ,
    emptyString

PROPERTIES
    noValue ∈ INT ∧
    noValue = − 1 ∧
    maxRef ∈ INT ∧
    maxRef = 100 ∧
    noReference ∈ INT ∧
    noReference = 0 ∧
    noQuantity ∈ INT ∧
    noQuantity = − 1 ∧
    noID ∈ INT ∧
    noID = − 1 ∧
    maxInt ∈ INT ∧
    maxInt = 65535 ∧
    emptyString ∈ String

ABSTRACT_VARIABLES
    values ,
    keys

INVARIANT
    values ∈ 1 .. maxRef → INT ∧
    keys ∈ ℙ ( 1 .. maxRef ) ∧
    ∀ Ref . ( Ref ∈ 1 .. maxRef ∧ Ref /∈ keys ⇒ ( values ( Ref ) = noValue ) ) ∧
    card ( keys ) ≤ maxRef ∧
    ∀ Ref . ( Ref ∈ 1 .. maxRef ∧ Ref ∈ keys ⇒ ( values ( Ref ) ≥ 0 ) )

INITIALISATION
    values := ( 1 .. maxRef ) × { noValue } ||
    keys := ∅

OPERATIONS
Result ⟵ getEntry ( key ) =
    PRE
        key ∈ INT ∧
        key ∈ keys
    THEN
        ANY
            l_values ,
            l_Result
        WHERE
            l_Result ∈ INT ∧
```

```
            l_values ∈ 1 .. maxRef → INT ∧
            Result = values ( key ) ∧
            l_values = values
        THEN
            values := l_values ||
            Result := l_Result
        END
    END ;
Result ⟵ deleteEntry ( key ) =
    PRE
        key ∈ INT ∧
        key ∈ keys
    THEN
        ANY
            l_keys , l_values ,
            l_Result
        WHERE
            l_Result ∈ 𝔹 ∧
            l_keys ⊆ 1 .. maxRef ∧
            keys = keys − { key } ∧
            l_values ∈ 1 .. maxRef → INT ∧
            values ( key ) = noValue ∧
             ( ∀ ii . ( ii ∈ 1 .. maxRef ∧ ( ii ≠ key ) ⇒ ( values ( ii ) = values ( ii ) ) ) )
        THEN
            keys := l_keys ||
            values := l_values ||
            Result := l_Result
        END
    END ;
Result ⟵ setEntry ( value , key ) =
    PRE
        value ∈ INT ∧
        key ∈ INT ∧
        ( 1 ≤ key ∧ key ≤ maxRef ) ∧
        value ≥ 0 ∧
        ( ¬ ( key ∈ keys ) ⇒ card ( keys ) < maxRef )
    THEN
        ANY
            l_values ,
            l_Result , l_keys
        WHERE
            l_Result ∈ INT ∧
            l_Result = key ∧
            l_keys ∈ 𝔽 ( 1 .. maxRef ) ∧
            l_values ∈ 1 .. maxRef → INT ∧
            l_keys = keys ∪ { key } ∧
            l_values = values ⩤ { ( key ↦ value ) }
        THEN
            values := l_values ||
            keys := l_keys ||
            Result := l_Result
        END
    END ;

Result ⟵ addToEntry ( value , key ) =
    PRE
        value ∈ INT ∧
        key ∈ INT ∧
        ( 1 ≤ key ∧ key ≤ maxRef ) ∧
        key ∈ keys ∧
        values ( key ) + value ∈ INT ∧
        values ( key ) + value ≥ 0
    THEN
        ANY
            l_Result , l_values
        WHERE
            l_Result ∈ INT ∧
            l_Result = key ∧
```

```
            l_values ∈ 1 .. maxRef → INT ∧
            l_values = values <+ { ( key ↦ values ( key ) + value ) }
        THEN
            Result := l_Result ||
            values := l_values
        END
    END ;
Result ⟵ getNewEntry =
    PRE
        card ( keys ) < maxRef
    THEN
        ANY
            l_Result
        WHERE
            l_Result ∈ 1 .. maxRef ∧
            l_Result /∈ keys
        THEN
            Result := l_Result
        END
    END
END
```

B.3 Manager_SetEntry_Store

```
/* SetEntry_Store
 × Author∈ Messabihi/Andre
 × Creation date∈ mar. juil . 13 2010
 */
MACHINE
    StockManager_addToEntry_store
CONCRETE_CONSTANTS
    /* origin variables */
        o_keys ,
        o_values ,
        o_result ,
        o_noValue ,
    /* promoted variables */
        p_catalog ,
        p_labels ,
        p_stock ,
        p_result ,
    /* origin parameters */
        o_key ,
        o_value ,
    /* promoted parameters */
        p_id ,
        p_qty ,
    /* updated variables*/
        new_o_values ,
        new_p_stock
PROPERTIES
    /* origin typing */
        o_keys ⊆ 1 .. 100 ∧
        o_values ∈ 1 .. 100 → INT ∧
        o_result ∈ INT ∧
        o_noValue ∈ INT ∧
        o_noValue = − 1 ∧
    /* promoted typing */
        p_catalog ⊆ 1 .. 100 ∧
        p_labels ∈ 1 .. 100 → INT ∧
        p_stock ∈ 1 .. 100 → INT ∧
        p_result ∈ INT ∧
    /* origin parameters typing */
        o_key ∈ INT ∧
        o_value ∈ INT ∧
    /* promoted parameters typing */
        p_id ∈ INT ∧
```

```
        p_qty ∈ INT ∧
    /* updates variables typing */
        new_o_values ∈ 1 .. 100 → INT ∧
        new_p_stock ∈ 1 .. 100 → INT ∧
    /* origin invariant already proved */
        card ( o_keys ) ≤ 100 ∧
    /* origin post */
        o_result = o_key ∧
        new_o_values ( o_key ) = o_values ( o_key ) + o_value ∧
    /* maping predicat */
        p_result = o_result ∧
        p_catalog = o_keys ∧
        p_stock = o_values ∧
        p_id = o_key ∧
        o_value = p_qty ∧
        new_o_values = new_p_stock ∧
    /* promoted pre */
        ( p_id ∈ p_catalog ∧ p_stock ( p_id ) + p_qty ≥ 0 ∧ p_qty > 0 )
ASSERTIONS
    /* origin pre */
        o_key ∈ o_keys ∧ o_values ( o_key ) + o_value ≥ 0 ∧
    /* promoted post */
        p_result = p_id ∧
        p_catalog = p_catalog ∪ { p_id } ∧
        new_p_stock ( p_id ) = p_stock ( p_id ) + p_qty ∧
        new_p_stock ( p_id ) > p_stock ( p_id )
END
```

C The AtelierB proof obligations

C.1 IntDictionary and StockManager_addToEntry_store

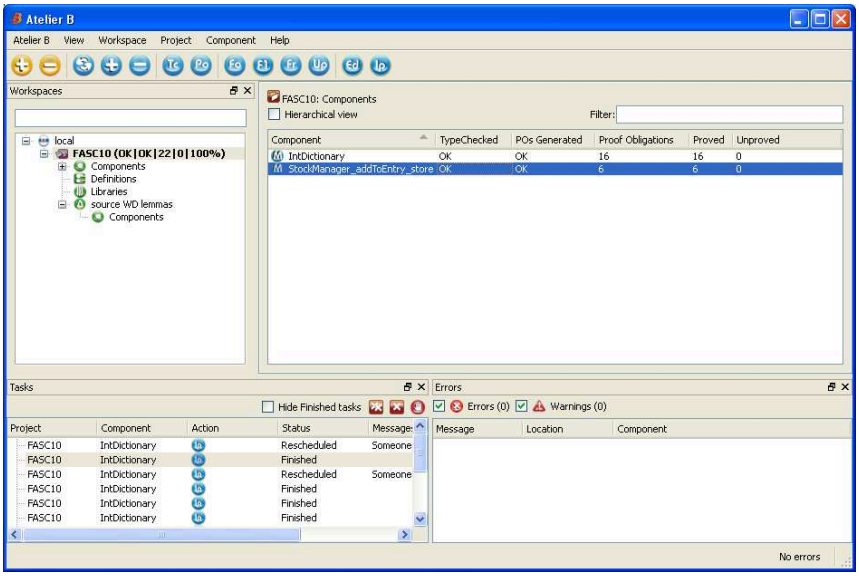

Fig. 5. Machine IntDictionary proved