Research Report

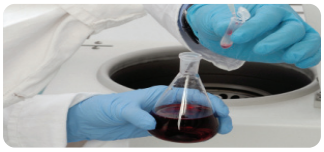
Departament de Llenguatges
i Sistemes Informàtics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

A basic set of test cases for a fragment of the osCommerce conceptual schema

Albert Tort
albert.tort@upc.edu

November, 2009

Table of contents



Validation through testing is commonly used in several scientific and industrial contexts.

1. Introduction

In several scientific and industrial contexts, such as medical research, civil engineering or aeronautics, testing is, clearly, a critical activity. Analyzing the resultant effects of applying our solutions in concrete situations is a common activity that helps **validating** the products developed by humans.

In the information systems development field, most research efforts has been devoted to code testing. But nowadays, most work in conceptual modeling assumes that conceptual schemas are executable and, consequently, they can also be tested. **Testing a conceptual schema** contributes to its validation early in the development, during the requirements engineering phase.

Conceptual schemas are the "general knowledge that an information system needs to know" [1]. We use UML/OCL [3,4] modeling languages to explicitly represent conceptual schemas. In contrast with a sequence of lines of code, conceptual schemas are represented by a set of conceptual elements (entity types, relationship types, integrity constraints, events, etc.). Therefore, there are important differences between testing code and testing conceptual schemas.

We reported in [7,8] **five kinds of tests** that are unique to conceptual schema testing:

- Asserting the consistency of an IB state.
- Asserting the inconsistency of an IB state.
- Asserting the occurrence of a domain event.
- Asserting the non-occurrence of a domain event.
- Asserting the contents of an IB state.

In [7,8] we also proposed a **Conceptual Schema Testing Language (CSTL)** that allows specifying this kinds of tests.

In the report [6] there are **many examples of test cases** to test the conceptual schema of the osCommerce system [5].

1.1. A Basic Set of Test Adequacy Criteria

A conceptual schema *CS* consists of a structural (sub)schema and a behavioral (sub)schema.

The structural subschema consists of a taxonomy of entity types $E = \{E_1,\ldots,E_n\}$, a set $R = \{R_1,\ldots,R_n\}$ of relationship types and a set *IC* of integrity constraints. Entity types and relationship types are types of the schema ($T = E \cup R$). Basic types can be based or derived ($T = T_{base} \cup T_{derived}$).

We admit multiple classification. Therefore, an entity may be an instance of one or more entity types. In a conceptual schema with multiple classification, an entity must be instance of a *valid type configuration* $VTC_i=\{E_1\ldots E_n\}$ [1, ch. 10]. *VTC* is the set of all the *valid type configurations* of the schema.

The behavioral subschema consists of a set *Dev* of domain event types. We model events as entities [2], which have characteristics, constraints and effects.

Given a test set *TS* of a conceptual schema *CS*, we can analyze whether *TS* is adequate or not according to a set of test adequacy criteria.

We propose **a basic set of four test adequacy criteria.**

All test sets of conceptual schemas should satisfy them in order to ensure that the **satisfiability of all the elements of the schema** has been thoroughly proved by testing. Ensuring the satisfiability of all the elements of the schema also implies that each element has been exercised in at least one test case.

A test set *TS* executes a set *TA* of one or more assertions TA_k. Analyzing whether *TS* satisfies the following criteria only makes sense if the verdict of all TA_k is true (all test cases pass).

Base Type Adequacy Criterion

A base type is satisfiable if it may have a non-empty population at certain time.

Let:

$BaseTypes(TA_k) = \{T_i \mid T_i \in T_{base}$ and there are one of more instances of T_i in at least one of the IB states found consistent during the evaluation of $TA_k\}$

$$BaseTypes(\mathrm{TA}) = \bigcup_{TA_k \in TA} BaseTypes(TA_k) \ .$$

Then, we say that:

A test set *TS* satisfies the *base type adequacy criterion* if and only if $T_{base} = BaseTypes(TA)$.

Derived Type Adequacy Criterion

A derived type is satisfiable if its derivation rule may derive at least one instance of it at a certain time.

Let:

$DerTypes(TA_k) = \{T_i \mid T_i \in T_{der}$ and the evaluation of TA_k in a state found consistent has required the derivation of one or more instances of $T_i\}$

$$DerTypes(TA) = \bigcup_{TA_k \in TA} DerTypes(TA_k)$$

Then, we say that:

A test set *TS* satisfies the *derived type adequacy criterion* if and only if T_{der} = DerTypes(TA).

Valid Type Configuration Adequacy Criterion

In multiple-classification models, the satisfiability property applies not only to the individual entity types, but also to the set of valid configurations of entity types.

Let:

$VTC(TA_k) = \{VTC_i \mid VTC_i \in VTC$ and there are one or more instances of VTC_i in at least one of the IB states found consistent during the evaluation of $TA_k\}$

$$VTC(TA) = \bigcup_{TA_k \in TA} VTC(TA_k)$$

Then, we say that:

A test set *TS* satisfies the v*alid type configuration adequacy criterion* if and only if *VTC = VTC(TA)*

Domain Event Type Adequacy Criterion

A domain event type Dev_i is satisfiable if there is at least one consistent state of the IB and one instance d of Dev_i with a set of characteristics such that the event constraints are satisfied, and the effects of d leave the IB in a state that is consistent and satisfies the event postconditions.

Let:

$DevType(TA_k) = \{Dev_i \mid Dev_i \in Dev$ and there is an instance of Dev_i the occurrence of which has been asserted by $TA_k\}$

$$DevType(TA) = \bigcup_{TA_k \in TA} DevType(TA_k)$$

Then, we say that:

A test set *TS* satisfies the *domain event type adequacy criterion* if and only if *Dev = DevType(TA)*

1.2. A basic set of test cases for a fragment of the osCommerce Conceptual Schema

In this document, we report a set of test cases *TS* of the conceptual schema fragment of a real information system for managing online stores. The set of test cases satisfy the *basic set of test adequacy criteria* explained above.

Given that the verdict of all the assertions of the *TS* is *Pass* and *TS* satisfies the *basic set of test adequacy criteria*, we can state that *TS* proves the satisfiability of all the elements of the conceptual schema.

The conceptual schema *CS* under test is a fragment of the osCommerce conceptual schema [5] that represents all the essential structural and behavioral knowledge needed to perform the main user functionalities of the osCommerce system when placing an order:

- Add products to a shopping cart when surfing the online store.
- Log in the system as a registered user.
- Update the shopping cart.
- Confirm an order.

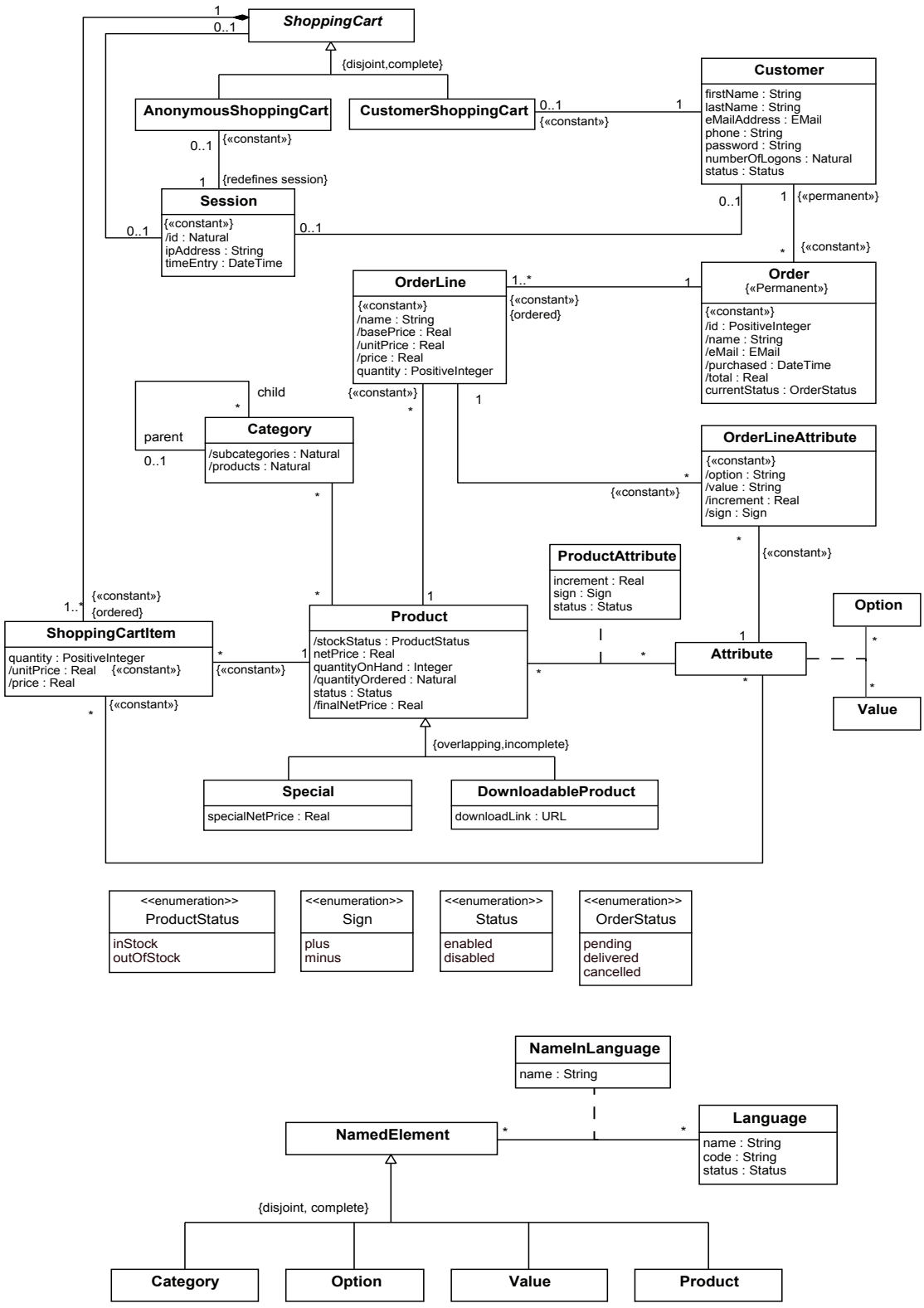
2. Fragment of the osCommerce Conceptual Schema

In this section, we present the fragment of the osCommerce conceptual schema focused on the structural and the behavioral knowledge needed to manage shopping carts and confirm orders.

Firstly, we present the (sub)structural schema, including the derivation rules for derived types and the integrity constraints expressed in OCL.

Secondly, we present the (sub)behavioral schema which consists of the set of domain event types modeled as entities. For each domain event type we specify its characteristics and constraints, the effect of its execution (in OCL) and its procedural method (using the CSTL language).

Structural schema



Derivation rules

***[DR1] Product::quantityOrdered** is the quantity of ordered items of this product.*

```
context Product::quantityOrdered(): Natural
  body : self.orderLine.quantity->sum()
```

***[DR2] Product::finalNetPrice** is the net price of the product considering if the product is an special.*

```
context Product::quantityOrdered(): Natural
  body : if self.oclIsTypeOf(Special) then self.oclAsType(Special).specialNetPrice else netPrice endif
```

***[DR3] Product::stockStatus** indicates whether there are product units to be sold or not*

```
context Product::stockStatus(): Natural
  body : if quantityOnHand>0 then ProductStatus::inStock else ProductStatus::outOfStock endif
```

***[DR4] Category::subcategories** is the number of subcategories owned by the category.*

```
context Category::subcategories(): NaturalI
 body :
        let allParents (): Set(Category) =
            if self.parent->notEmpty() then self.parent -> union(self.parent.allParents()) else Set{} endif
        in
          Category.allInstances() -> select(c| c.allParents()-> includes(self))->size()
```

***[DR5] Category::products** is the number of products owned by the category.*

```
context Category::products(): Natural
 body :
        let allParents() : Set(Category) =
            if self.parent->notEmpty() then self.parent -> union(self.parent.allParents()) else Set{} endif
        in
          Category.allInstances() -> select(c | c.allParents() -> includes(self) or c=self).product->size()
```

***[DR6] ShoppingCartItem::unitPrice** is the unit price of the product of the shopping cart item taking into account the selected product attributes.*

```
context ShoppingCartItem::unitPrice():Real
 body :
     self.attribute.productAttribute -> select (pa | pa.product = self.product) -> collect
     (if sign = Sign::plus
     then increment
      else –increment
     endif) -> sum() + self.product.finalNetPrice
  endif
```

***[DR7] ShoppingCartItem::price** is the price of the shopping cart item taking into account the quantity and the selected product attributes.*

```
context ShoppingCartItem::price():Real
 body :  self.unitPrice * self.quantity
```

***[DR8] Order::id** identifies the order and it is automatically derived.*

```
context Order::id():PositiveInteger
 body :
  if Order.allInstances() -> size() = 1 then 1
  else (Order.allInstances()->excluding(self)) -> sortedBy(id) -> last().id + 1
  endif
```

***[DR9] Order::eMail** of an order is that of its customer.*

```
context Order::eMail ():EMail
  body : self.customer.eMailAddress
```

***[DR10] Order::purchased** is the DateTime when the order was created*

```
context Order::purchased():DateTime
  body : Now()
```

***[DR11] Order::total** gives the total amount of an order*

```
context Order::total():Real
  body :   self.orderLine.price-> sum()
```

***[DR12] Order::name** is the customer name of the order.*

```
context Order::name():String
body :  self.customer.firstName.concat(" ").concat(self.customer.lastName)
```

***[DR13] OrderLine::basePrice** is the price of the product of the order line without taking into account the purchased quantity and the selected attributes.*

```
context OrderLine::basePrice():Real
  body :  self.product.netPrice
```

***[DR14] OrderLine::unitPrice** is the price of one item of the product of the order line taking into account the selected attributes.*

```
context OrderLine::price():Real
body :
      self.orderLineAttribute -> collect
      (if sign = Sign::plus then increment
      else –increment
      endif) -> sum() + self.basePrice
   endif
```

***[DR15] OrderLine::price** is the final price of the items of the order line.*

```
context OrderLine::finalPrice():Real
  body :  self.unitPrice * self.quantity
```

***[DR16] OrderLine::name** is the name of the product of the order line.*

```
context OrderLine::name():String
  body :  self.product.nameInLanguage->any(true).name
```

***[DR17] OrderLineAttribute::option** is the name of the option of the order line attribute.*

```
context OrderLineAttribute::option():String
  body :  self.attribute.option.nameInLanguage->any(true).name
```

***[DR18] OrderLineAttribute::value** is the name of the value of the order line attribute.*

```
context OrderLineAttribute::value():String
  body :  self.attribute.value.nameInLanguage->any(true).name
```

***[DR19] OrderLineAttribute::increment** is the increment applied in the product price by the attribute of the order line attribute.*

```
context OrderLineAttribute::increment():Real
body :
   self.attribute.productAttribute
      -> select (pa | pa.product = self.orderLine.product)->any(true).increment
```

***[DR20] OrderLineAttribute::sign** is the sign of the increment applied in the product price by the attribute of the order line attribute.*

```
context OrderLineAttribute::sign():Sign
body :
   self.attribute.productAttribute
   -> select (pa | pa.product = self.orderLine.product)->any(true).sign
```

***[DR21] Session::id** is the identifier of the session.*

```
context Session::id():Natural
body :  Session.allInstances()->size()
```

Integrity Constraints

[IC1] A language is identified by its name and by its code

```
context Language::codeAndNameAreUnique: Boolean
  body : Language.allInstances() -> isUnique(name) and  Language.allInstances() -> isUnique(code)
```

[IC2] Named elements are identified by its name in each language.

```
context NamedElement::nameIsUnique(): Boolean
  body : self.language->forAll(nameInLanguage->isUnique(name))
```

[IC3] Named elements must have a name in each language.

```
context NamedElement::aNameInEachLanguage(): Boolean
  body :  self.language = Language.allInstances()
```

[IC4] There are no cycles in category hierarchies.

```
context Category::isAHierarchy(): Boolean
  body : not self.allParents() -> includes(self)
```

[IC5] Customers are identified by their email address.

```
context Customer::eMailIsUnique(): Boolean
  body : Customer.allInstances() -> isUnique(eMailAddress)
```

[IC6] Sessions are identified by its id.

```
context Session::idIsUnique(): Boolean
  body : Session.allInstances() -> isUnique (id)
```

[IC7] If a customer shopping cart exists in the context of a session then its customer is the customer of the session.

```
context CustomerShoppingCart::sameCustomer(): Boolean
  body : self.session.customer -> notEmpty()  implies  self.session.customer = self.customer
```

[IC8] The set of attributes of a shopping cart item must be attributes of the shopping cart item product.

```
context ShoppingCartItem::productHasTheAttributes(): Boolean
  body : self.product.attribute -> includesAll(self.attribute)
```

[IC9] The shopping cart item specifies only one attribute per option.

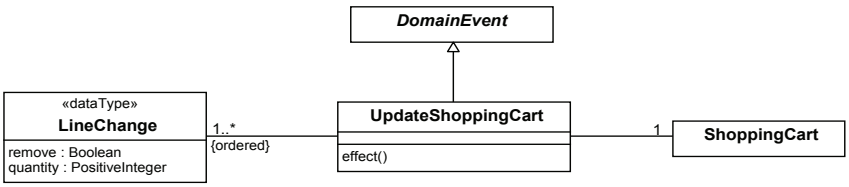
```
context ShoppingCartItem::onlyOneAttributePerOption(): Boolean
  body : self.attribute -> isUnique(option)
```

[IC10] Orders are identified by its id

```
context Order::idIsUnique: Boolean
body :  Order.allInstances() -> isUnique(id)
```

Behavioral Schema

UpdateShoppingCart



[Initial Integrity Constraints]
Line change modifications must be complete according to the shopping cart items of the shopping cart

```
context UpdateShoppingCart::complete(): Boolean
body :  self.lineChange->size() = self.shoppingCart.shoppingCartItem->size()
```

Event effect

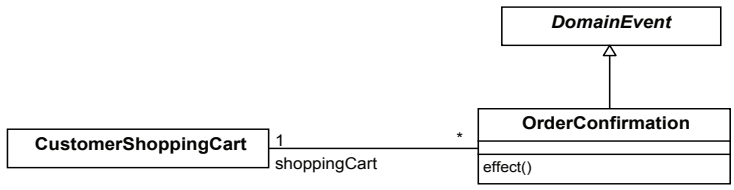
```
context UpdateShoppingCart::effect()
post :
   self.lineChange -> forAll
     (lc | let cartItem:ShoppingCartItem =
                self.shoppingCart.shoppingCartItem@pre -> at(lineChange->indexOf(lc))
           in
                (lc.remove or lc.quantity <> cartItem.quantity)
                 implies
                if lc.remove then
                  not cartItem@pre.oclIsKindOf(OclAny)
                 else
                   cartItem.quantity = lc.quantity
                 endif )
```

Event method

```
method UpdateShoppingCart::effect(){
i:=1;
iItems:=1;
cartItems := self.shoppingCart.shoppingCartItem;
```

```
while self.lineChange->size()>=i do
      lc := self.lineChange->at(i);
      cartItem := cartItems->at(i);
      if lc.remove then
         delete cartItem;
      else
         cartItem.quantity := lc.quantity;
         iItems:=iItems+1;
      endif
       i:=i+1;
      endwhile
}
```

OrderConfirmation



Event effect

```
context OrderConfirmation::effect()
post :
    (Order.allInstances() - Order.allInstances()@pre) -> one(o:Order |
     o.oclIsNew() and
     o.oclIsTypeOf(Order) and
     o.customer = self.shoppingCart@pre.customer@pre and
     --The initial status of the order
     o.currentStatus = Status::pending and
     --There is an order line for each shopping cart item
    shoppingCart@pre.shoppingCartItem@pre->forAll(i|OrderLine.allInstances() -> one
        (ol|ol.order = o  and
            ol.product = i.product@pre  and
            ol.quantity = i.quantity@pre  and
            i.attribute@pre->forAll
               (iAtt|OrderLineAttribute.allInstances -> exists
               (olAtt|olAtt.orderLine = ol and
                     olAtt.attribute = iAtt)))))
  post theShoppingCartIsRemoved:
    ShoppingCart.allInstances->excludes(self.shoppingCart@pre)
  post updateProductQuantities:
    let productsBought:Set(Product) =
       self.shoppingCart@pre.shoppingCartItem@pre.product@pre->asSet()
    in  productsBought -> forAll (p|
          let quantityBought:Integer =
               self.shoppingCart@pre.shoppingCartItem@pre->select
                 (sc | sc.product = p).quantity -> sum()
          in
               p.quantityOnHand = p.quantityOnHand@pre - quantityBought)
```

Event method

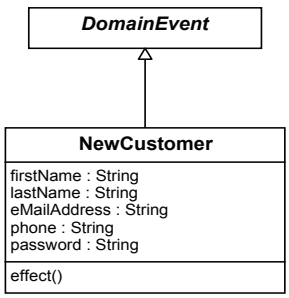
```
method OrderConfirmation::effect(){
//The order is created
o:=new Order;
o.customer := self.shoppingCart.customer;
//The initial status of the order
o.currentStatus := #pending;
```

```
//There is an order line for each shopping cart item
 index:=0;
indexat:=0;
while self.shoppingCart.shoppingCartItem->size()>index do
   sci := self.shoppingCart.shoppingCartItem->at(index+1);
   ol:=new OrderLine;
   ol.order:=o;
   ol.product:=sci.product;
   ol.quantity:=sci.quantity;
   while sci.attribute->size()>indexat do
      attr:=sci.attribute->asSequence()->at(indexat+1);
      ola:=new OrderLineAttribute;
      ola.orderLine:=ol;
      ola.attribute:=attr;
      indexat:=indexat+1;
   endwhile
      index:=index+1;
      indexat:=0;
 endwhile

//update product quantities
products:=o.orderLine.product->asSet();
j:=0;
while products->size()>j do
    p:=products->asSequence()->at(j+1);
    var:=o.orderLine->select(product=p).quantity->sum();
    p.quantityOnHand:=p.quantityOnHand-var;
    j:=j+1;
 endwhile
self.createdOrder:=o;

//The shopping cart is removed
while self.shoppingCart.shoppingCartItem->size()>0 do
    z:=self.shoppingCart.shoppingCartItem->any(true);
    self.shoppingCart.shoppingCartItem:=self.shoppingCart.shoppingCartItem->excluding(z);
    delete z;
 endwhile
delete self.shoppingCart;
}
```

NewCustomer



[Initial Integrity Constraints]
The customer does not exist

```
context NewCustomer::customerDoesNotExist(): Boolean
body :  not Customer.allInstances() -> exists (c | c.eMailAddress=self.eMailAddress)
```

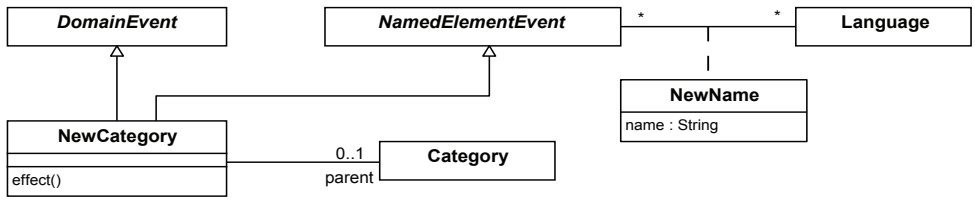
Event effect

```
context NewCustomer::effect()
post :
  (Customer.allInstances() - Customer.allInstances()@pre) -> one(c:Customer |
     c.oclIsNew() and
     c.oclIsTypeOf(Customer) and
     c.firstName = self.firstName and
     c.lastName = self.lastName and
     c.eMailAddress = self.eMailAddress and
     c.phone = self.phone and
     c.password = self.password and
     c.numberOfLogons = 0 and
     c.status =Status::enabled)
```

Event method

```
method NewCustomer::effect(){
c:=new Customer;
c.firstName:=self.firstName;
c.lastName:=self.lastName;
c.eMailAddress:=self.eMailAddress;
c.phone:=self.phone;
c.password:=self.password;
c.numberOfLogons:=0;
c.status:=#enabled;
self.createdCustomer:=c;
}
```

NewCategory



[Initial Integrity Constraints]
A name in each language must be specified

```
context NamedElementEvent::aNameInEachLanguage(): Boolean
body :  Language.allInstances() = self.language
```

The category does not exist

```
context NewCategory::categoryDoesNotExist(): Boolean
body :  not Category.allInstances()->exists(c | c.nameInLanguage.name = self.newName.name)
```

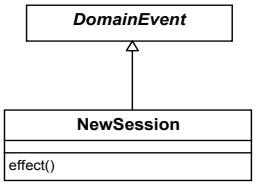
Event effect

```
context NewCategory::effect()
post :
    (Category.allInstances() – Category.allInstances()@pre) -> one(c:Category |
     c.oclIsNew()  and
     c.oclIsTypeOf(Category) and
     c.parent = self.parent and
     self.newName.name=c.nameInLanguage.name)
```

Event method

```
method NewCategory::effect(){
cat:=new Category;
cat.parent:=self.parent;
i:=0;
while Language.allInstances->size()>i do
   l:=Language.allInstances->asSequence()->at(i+1);
   catInLanguage:=self.newName->select(language=l)->any(true);
   cil:=new NameInLanguage(namedElement:=cat,language:=l);
   cil.name:=catInLanguage.name;
    i:=i+1;
 endwhile
self.createdCategory:=cat;
}
```

NewSession



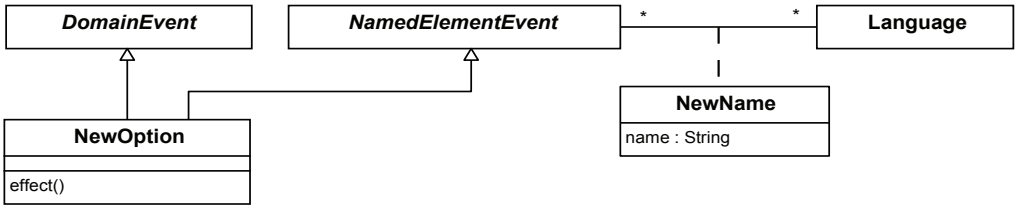
Event effect

```
context NewSession::effect()
post :
     (Session.allInstances - Session.allInstances@pre) ->one(s:Session |
      s.oclIsNew() and
      s.oclIsTypeOf(Session) and
      s.ipAddress=IPAddress() and
      s.timeEntry=Now() )
```

Event method

```
method NewSession::effect(){
s:=new Session;
self.createdSession:=s;
s.timeEntry:=Now()';
s.ipAddress:=IPAddress();
}
```

NewOption



[Initial Integrity Constraints]
The option does not exist

```
context NewOption::optionDoesNotExist(): Boolean
body :  not Option.allInstances()->exists(o | o.nameInLanguage.name = self.newName.name)
```

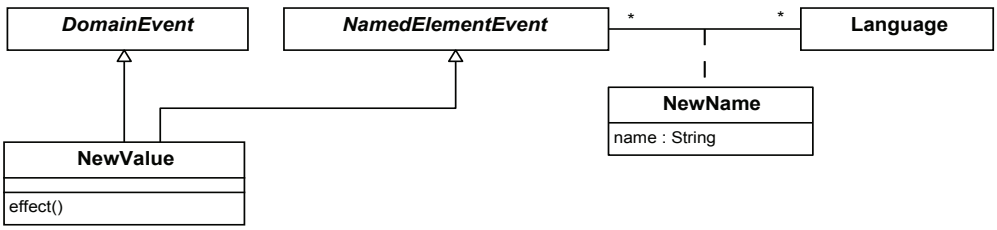
Event effect

```
context NewOption::effect()
post :
  (Option.allInstances() - Option.allInstances()@pre) -> one(o:Option |
      o.oclIsNew()  and
      o.oclIsTypeOf(Option) and
      self.newName.name=o.nameInLanguage.name)
```

Event method

```
method NewOption::effect(){
op:=new Option;
i:=0;
while Language.allInstances->size()>i do
   l:=Language.allInstances->asSequence()->at(i+1);
   optInLanguage:=self.newName->select(language=l)->any(true);
   oil:=new NameInLanguage(namedElement:=op,language:=l);
   oil.name:=optInLanguage.name;
    i:=i+1;
 endwhile
self.createdOption:=op;
}
```

NewValue



[Initial Integrity Constraints]
The value does not exist

```
context NewValue::valueDoesNotExist(): Boolean
body :  not Value.allInstances()->exists(v | v.nameInLanguage.name = self.newName.name)
```

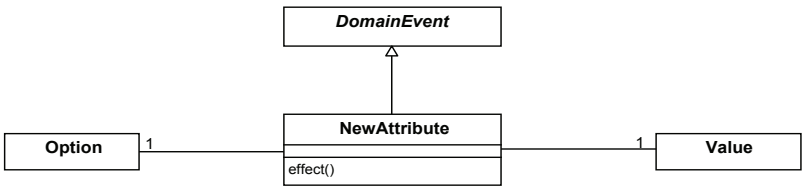
Event effect

```
context NewValue::effect()
post :
  (Value.allInstances() - Value.allInstances()@pre) -> one(v: Value |
      v.oclIsNew()  and
      v.oclIsTypeOf(Value) and
      self.newName.name=o.nameInLanguage.name)
```

Event method

```
method NewValue::effect(){
v:=new Value;
i:=0;
while Language.allInstances->size()>i do
   l:=Language.allInstances->asSequence()->at(i+1);
   valInLanguage:=self.newName->select(language=l)->any(true);
   vil:=new NameInLanguage(namedElement:=v,language:=l);
   vil.name:=valInLanguage.name;
    i:=i+1;
 endwhile
self.createdValue:=v;
}
```

NewAttribute



[Initial Integrity Constraints]
The attribute does not exist

```
context NewAttribute:: AttributeDoesNotExist(): Boolean
body :  not Attribute.allInstances() -> exists(a | a.value=self.value  and  a.option = self.option)
```

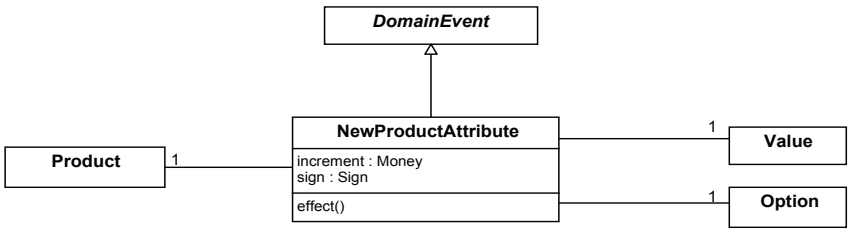
Event effect

```
context NewProductAttribute::effect()
post :
  (Attribute.allInstances() - Attribute.allInstances()@pre) -> one(a:Attribute |
    a.oclIsNew()  and
    a.oclIsTypeOf(Attribute) and
    a.option = self.option and
    a.value = self.value)
```

Event method

```
method NewAttribute::effect(){
attrValue:=self.value;
attrOption:=self.option;
a := new Attribute(option:=attrOption, value:=attrValue);
self.createdAttribute:=a;
}
```

NewProductAttribute



[Initial Integrity Constraints]
The product attribute does not exist

context NewProductAttribute::productAttributeDoesNotExist(): Boolean
body : **not** self.product.productAttribute -> exists(attribute.value=self.value **and** attribute.option = self.option)

The option-value pair is valid

context NewProductAttribute::optionValueIsValid(): Boolean
body : self.option.value -> includes(self.value)

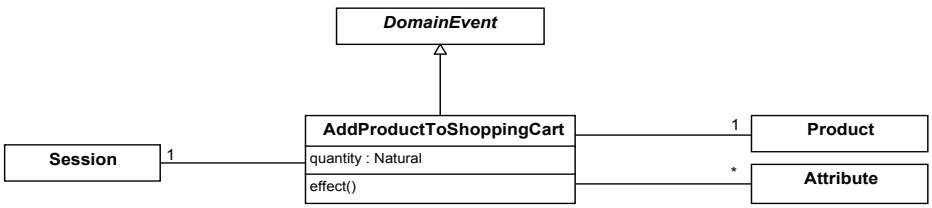
Event effect

context NewProductAttribute::effect()
post :
(ProductAttribute.allInstances() - ProductAttribute.allInstances()@pre) -> one(pa:ProductAttribute |
pa.oclIsNew() **and**
pa.oclIsTypeOf(ProductAttribute) **and**
pa.increment = self.increment **and**
pa.sign = self.sign **and**
pa.product = self.product **and**
pa.attribute.option = self.option **and**
pa.attribute.value = self.value **and**
pa.status = Status::enabled)

Event method

method NewProductAttribute::effect(){
o:=self.option;
v:=self.value;
attr:=Attribute.allInstances->select(value=v)->any(option=o);
pa:=**new** ProductAttribute(product:=self.product, attribute:=attr);
pa.sign:=self.sign;
pa.increment:=self.increment;
pa.status:=#enabled;
self.createdProductAttribute:=pa;
}

AddProductToShoppingCart



[Initial Integrity Constraints]
The attributes must be valid for the added product

```
context AddProductToShoppingCart::attributesAreFromProduct(): Boolean
body : self.product.attribute -> includesAll(self.attribute)
```

Only one value per option is allowed

```
context AddProductToShoppingCart:AttributesAreOfDifferentOptions(): Boolean
body : self.attribute -> isUnique(option)
```

Event effect

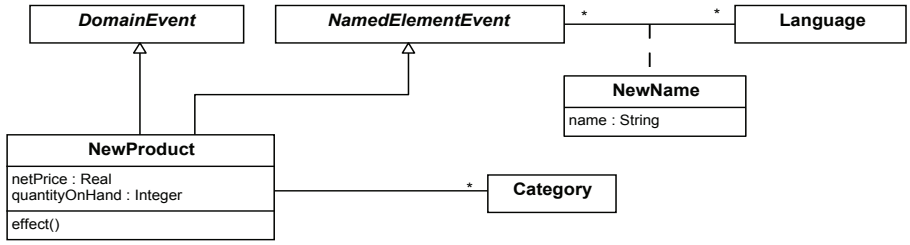
```
context AddProductToShoppingCart::effect()
post :
(ShoppingCartItem.allInstances() - ShoppingCartItem.allInstances()@pre) -> one(sci:ShoppingCartItem |
    sci.oclIsNew and
    sci.oclIsTypeOf(ShoppingCartItem) and
    sci.quantity = self.quantity and
    sci.product = self.product and
    sci.attribute = self.attribute and
    if self.session.shoppingCart -> notEmpty() then
      --The session has a shopping cart
      self.session.shoppingCart.shoppingCartItem -> includes(sci)
    else
      --The session does not have a shopping cart
      if self.session.customer -> isEmpty() then
      --The session is Anonymous
          (AnonymousShoppingCart.allInstances() - AnonymousShoppingCart.allInstances()@pre)
        -> one(sc:AnonymousShoppingCart |
        sc.oclIsNew()  and
        sc.oclIsTypeOf(AnonymousShoppingCart)  and
        self.session.shoppingCart = sc and
        self.session.anonymousShoppingCart = sc and
        sc.shoppingCartItem -> includes(sci))
      else
      --The customer is logged in
        if self.session.customer.customerShoppingCart -> notEmpty()  then
          --The customer has a previous shopping cart
          self.session.shoppingCart = self.session.customer.customerShoppingCart and
          self.session.shoppingCart.shoppingCartItem -> includes(sci)
        else
          --The customer does not have a previous shopping cart
          (CustomerShoppingCart.allInstances() - CustomerShoppingCart.allInstances()@pre)
          -> one(csc:CustomerShoppingCart |
          csc.oclIsNew()  and
          csc.oclIsTypeOf(CustomerShoppingCart)  and
          self.session.shoppingCart = csc and
          csc.shoppingCartItem -> includes(sci))
        endif
      endif
    endif)
```

Event method

```
method AddProductToShoppingCart::effect(){
sci:=new ShoppingCartItem;
sci.quantity:=self.quantity;
sci.product:=self.product;
sci.attribute:=self.attribute;
if self.session.shoppingCart->size()>0 then
  //The session has a shopping cart
  self.session.shoppingCart.shoppingCartItem := self.session.shoppingCart.shoppingCartItem->append(sci);
```

```
else
//The session does not have a shopping cart
   if self.session.customer.isUndefined() then
      //The session is anonymous
      asc := new AnonymousShoppingCart;
      self.session.shoppingCart:=asc;
      self.session.anonymousShoppingCart:=asc;
      asc.shoppingCartItem:=Sequence{sci};
   else
      //The customer is logged in
      if self.session.customer.customerShoppingCart->size()>0 then
         //The customer has a previous shopping cart
         self.session.customer.customerShoppingCart.shoppingCartItem :=
         self.session.customer.customerShoppingCart.shoppingCartItem->append(sci);
      else
         //The customer does not have a previous shopping cart
         csc:=new CustomerShoppingCart;
         csc.customer:=self.session.customer;
         csc.shoppingCartItem:=self.session.shoppingCart.shoppingCartItem;
         self.session.shoppingCart:=csc;
         csc.shoppingCartItem:=sci;
      endif
   endif
endif
}
```

NewProduct



[Initial Integrity Constraints]
The product does not exist

```
context NewProduct::productDoesNotExist(): Boolean
body :  not Product.allInstances()->exists(p | p.nameInLanguage.name = self.newName.name)
```

Event effect

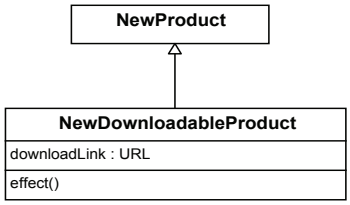
```
context NewProduct::effect()
post :
(Product.allInstances() - Product.allInstances()@pre) -> one(p:Product |
      p.oclIsNew()  and
      p.oclIsTypeOf(Product) and
      p.netPrice = self.netPrice and
      p.quantityOnHand = self.quantityOnHand and
      p.status = Status::enabled and
      p.category = self.category and
      self.newName.name = p.nameInLanguage.name)
```

Event method

```
method NewProduct::effect(){
p:=new Product;
p.status := #enabled;
```

```
p.netPrice:= self.netPrice;
p.quantityOnHand := self.quantityOnHand;
p.category := self.category;
i:=0;
while Language.allInstances->size()>i do
   l:=Language.allInstances->asSequence()->at(i+1);
   prodInLanguage:=self.newName->select(language=l)->any(true);
   pil:=new NameInLanguage(namedElement:=p,language:=l);
   pil.name:=prodInLanguage.name;
    i:=i+1;
 endwhile
self.createdProduct:=p;
}
```

NewDownloadableProduct



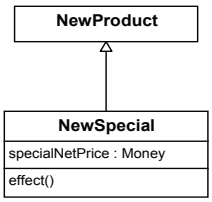
Event effect

```
context NewDownloadableProduct::effect()
post :
(DownloadableProduct.allInstances() - DownloadableProduct.allInstances()@pre)
      -> one(dp:DownloadableProduct |
     dp.oclIsNew()  and
     dp.oclIsTypeOf(DownloadableProduct) and
  dp.downloadLink = self.downloadLink)
```

Event method

```
method NewDownloadableProduct::effect(){
dp:=new DownloadableProduct;
dp.status := #enabled;
dp.netPrice:= self.netPrice;
dp.quantityOnHand := self.quantityOnHand;
dp.category := self.category;
dp.downloadLink:=self.downloadLink;
i:=0;
while Language.allInstances->size()>i do
   l:=Language.allInstances->asSequence()->at(i+1);
   prodInLanguage:=self.newName->select(language=l)->any(true);
   pil:=new NameInLanguage(namedElement:=dp,language:=l);
   pil.name:=prodInLanguage.name;
   i:=i+1;
 endwhile
self.createdDownloadableProduct:=dp;
self.createdProduct:=dp;
}
```

NewSpecial



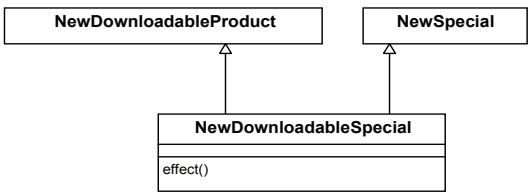
Event effect

```
context NewSpecial::effect()
post :
  (Special.allInstances() - Special.allInstances()@pre) -> one(s:Special |
     s.oclIsNew()  and
     s.oclIsTypeOf(Special) and
     s.specialNetPrice = self.specialNetPrice)
```

Event method

```
method NewSpecial::effect(){
s:=new Special;
s.status := #enabled;
s.netPrice:= self.netPrice;
s.quantityOnHand := self.quantityOnHand;
s.category := self.category;
s.specialNetPrice:=self.specialNetPrice;
i:=0;
while Language.allInstances->size()>i do
   l:=Language.allInstances->asSequence()->at(i+1);
   prodInLanguage:=self.newName->select(language=l)->any(true);
   pil:=new NameInLanguage(namedElement:=s,language:=l);
   pil.name:=prodInLanguage.name;
   i:=i+1;
 endwhile
self.createdSpecial:=s;
self.createdProduct:=s;
}
```

NewDownloadableSpecial



Event effect

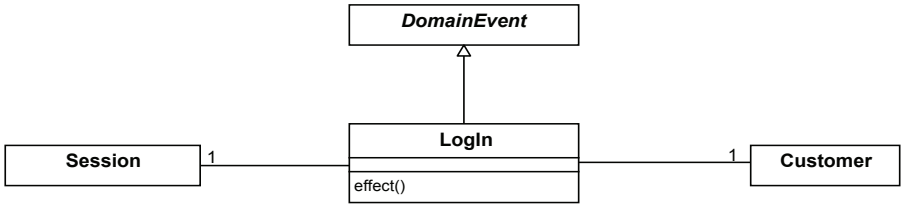
```
context NewDownloadableSpecial::effect()
post :  true
```

Event method

```
method NewDownloadableSpecial::effect(){
ds:=new DownloadableProduct,Special;
ds.status := #enabled;
```

```
ds.netPrice:= self.netPrice;
ds.quantityOnHand := self.quantityOnHand;
ds.category := self.category;
ds.specialNetPrice:=self.specialNetPrice;
ds.downloadLink:=self.downloadLink;
i:=0;
while Language.allInstances->size()>i do
   l:=Language.allInstances->asSequence()->at(i+1);
   prodInLanguage:=self.newName->select(language=l)->any(true);
   pil:=new NameInLanguage(namedElement:=ds,language:=l);
   pil.name:=prodInLanguage.name;
   i:=i+1;
 endwhile
self.createdDownloadableProduct:=ds;
self.createdSpecial:=ds;
self.createdProduct:=ds;
}
```

LogIn



[Initial Integrity Constraints]
The customer is not logged in

```
context LogIn::customerIsNotLoggedIn(): Boolean
body :  self.customer.session -> isEmpty()
```

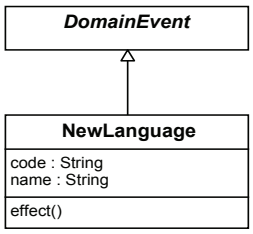
Event effect

```
context LogIn::effect()
  post IdentifySession:
     self.session.customer = self.customer
  post UpdateNumberOfLogons:
     self.customer.numberOfLogons = self.customer.numberOfLogons@pre + 1
  post RestorePreviousShoppingCart:
            let previousShoppingCart:CustomerShoppingCart = self.customer.customerShoppingCart
            in
            self.customer.customerShoppingCart->notEmpty() implies
            (self.session.shoppingCart=previousShoppingCart and
             previousShoppingCart.shoppingCartItem
                  ->includesAll(self.session.shoppingCart.shoppingCartItem) and
            previousShoppingCart.customer=self.customer and
            self.session.shoppingCart=previousShoppingCart)
  post AddAnonymousItems:
                  let anonymousShoppingCart:AnonymousShoppingCart =
                  self.session.anonymousShoppingCart
                  in
                  self.session.anonymousShoppingCart->notEmpty() implies
                     (let currentCustomerCart:ShoppingCart = self.session.shoppingCart
                        in
                         self.session.shoppingCart->notEmpty() and
                         currentCustomerCart.oclIsTypeOf(CustomerShoppingCart) and
                         currentCustomerCart.oclAsType(CustomerShoppingCart).customer=self.customer and
                         currentCustomerCart.shoppingCartItem
                               ->includesAll(anonymousShoppingCart.shoppingCartItem))
```

Event method

```
method LogIn::effect(){
s:=self.session;
s.customer := self.customer;
self.customer.numberOfLogons:=self.customer.numberOfLogons+1;
previousCustomerShoppingCart:=c.customerShoppingCart;
if c.customerShoppingCart->size()>0 then
   s.shoppingCart:=previousCustomerShoppingCart;
endif
if self.session.shoppingCart->size()=1 then
   if c.customerShoppingCart->size()>0 then
      s.shoppingCart:=previousCustomerShoppingCart;
      previousCustomerShoppingCart.shoppingCartItem:=self.session.shoppingCart.shoppingCartItem;
      self.session.shoppingCart.shoppingCartItem:=oclEmpty(Set(ShoppingCartItem));
      asc:=self.session.shoppingCart;
      delete asc;
   else
      csc:=new CustomerShoppingCart;
      csc.customer:=self.customer;
      csc.shoppingCartItem:=self.session.shoppingCart.shoppingCartItem;
      self.session.shoppingCart.shoppingCartItem:=oclEmpty(Set(ShoppingCartItem));
      asc:=self.session.shoppingCart;
      self.session.shoppingCart:=oclEmpty(Set(ShoppingCart));
      s.shoppingCart:=csc;
      delete asc;
   endif
else
   s.shoppingCart:=previousCustomerShoppingCart;
endif
}
```

NewLanguage



[Initial Integrity Constraints]
The language does not exist

```
context NewLanguage::languageDoesNotExist(): Boolean
body :  not Language.allInstances()->exists(l | l.code = self.code or l.name = self.name)
```

Event effect

```
context NewLanguage::effect()
  post:
   (Language.allInstances() - Language.allInstances()@pre) -> one(l:Language |
      l.oclIsNew()  and
      l.oclIsTypeOf(Language) and
      l.code = self.code and
      l.name = self.name and
      l.status = Status::enabled)
```

Event method

```
method NewLanguage::effect(){
l:=new Language;
self.createdLanguage:=l;
l.code:=self.code;
l.name:=self.name;
l.status:=#enabled;
}
```

3. Basic set of tests

In this section we report test set of the fragment of the osCommerce conceptual schema presented in Section 2. **The test set satisfies the *basic set of test adequacy criteria*** explained in Section 1.

We extended our CSTL test processor prototype in order to **automatically perform the analysis of coverage** according to the *basic set of test adequacy criteria*.

We present the basic set of tests in five progressive steps. In each step we add some test cases to increase the fulfillment of the adequacy criteria. In the last step we reach a set of test cases that satisfy the basic adequacy criteria at all.

As an starting point, if we perform the coverage analysis with an empty set of test cases, the informative results given by the test processor (Figure 1) indicate that all the elements are uncovered according to the *basic set of test adequacy criteria*.

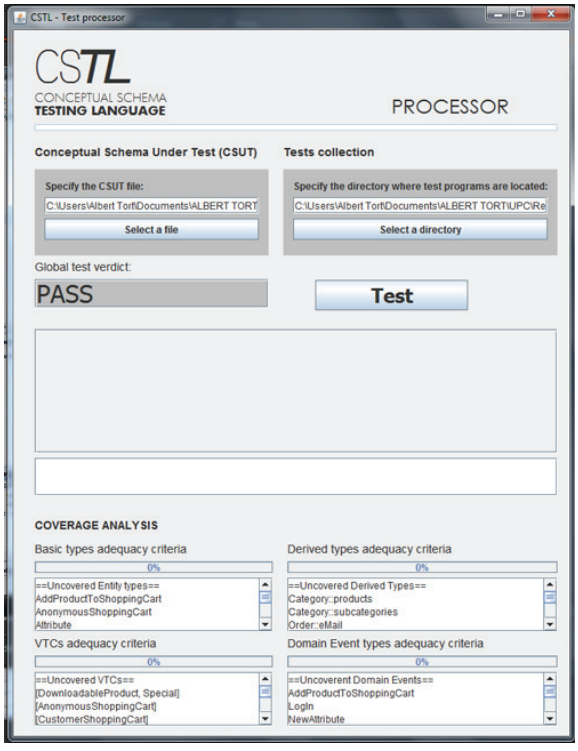


Figure 1. Coverage analysis for an empty set of tests.

Uncovered elements report

==Uncovered Entity types==
AddProductToShoppingCart
AnonymousShoppingCart
Attribute
Category
Customer
CustomerShoppingCart
DownloadableProduct
Language
LineChange
LogIn
NameInLanguage
NamedElement
NewAttribute
NewCategory
NewCustomer
NewDownloadableProduct
NewDownloadableSpecial
NewLanguage
NewName
NewOption
NewProduct
NewProductAttribute
NewSession
NewSpecial
NewValue
Option
Order
OrderConfirmation
OrderLine
OrderLineAttribute
Product
ProductAttribute
Session
ShoppingCart
ShoppingCartItem
Special
UpdateShoppingCart
Value

==Uncovered Relationship types==
Attribute
NameInLanguage
NewName
ProductAttribute
addProductToShoppingCart_attribute
addProductToShoppingCart_product
addProductToShoppingCart_session
anonymousShoppingCart_session
category_product
customerShoppingCart_customer
customer_order
logIn_customer
logIn_session
newAttribute_option
newAttribute_value
newCategory_category
newProductAttribute_option
newProductAttribute_product
newProductAttribute_value
newProduct_category
orderConfirmation_customerShoppingC
art
orderLineAttribute_attribute
orderLine_orderLineAttribute
orderLine_product
order_orderLine
parent_child
session_customer
shoppingCartItem_attribute
shoppingCartItem_product
shoppingCart_session
shoppingCart_shoppingCartItem
updateShoppingCart_lineChange
updateShoppingCart_shoppingCart

==Uncovered Derived Types==
Category::products
Category::subcategories
Order::eMail
Order::id
Order::name
Order::purchased
Order::total
OrderLine::basePrice
OrderLine::name
OrderLine::price
OrderLine::unitPrice
OrderLineAttribute::increment
OrderLineAttribute::option
OrderLineAttribute::sign
OrderLineAttribute::value
Product::finalNetPrice
Product::quantityOrdered
Product::stockStatus
Session::sessionID
ShoppingCartItem::price
ShoppingCartItem::unitPrice

==Uncovered VTCs==
[DownloadableProduct, Special]
[AnonymousShoppingCart]
[CustomerShoppingCart]
[Session]
[Customer]
[OrderLine]
[Category]
[Category]
[Order]
[OrderLineAttribute]
[Product]
[ShoppingCartItem]
[Attribute]
[Option]
[Value]
[ProductAttribute]
[DownloadableProduct]
[Special]
[Language]

==Uncoverent Domain Events==
AddProductToShoppingCart
LogIn
NewAttribute
NewCategory
NewCustomer
NewDownloadableProduct
NewDownloadableSpecial
NewLanguage
NewOption
NewProduct
NewProductAttribute
NewSession
NewSpecial
NewValue
OrderConfirmation
UpdateShoppingCart

Now, consider the following test program:

```
testprogram PlaceAnOrderProcess{

nlang := new NewLanguage(code:='en', name:='english');
assert occurrence nlang;
english:=nlang.createdLanguage;

ns := new NewSession;
assert occurrence ns;

np1 := new NewProduct(netPrice:=3, quantityOnHand:=50);
nn:= new NewName(namedElementEvent:=np1, language:=english);
nn.name:='BarcelonaMap';
assert occurrence np1;
barcelonaMap:=np1.createdProduct;


np2 := new NewProduct(netPrice:=30, quantityOnHand:=5);
nn:= new NewName(namedElementEvent:=np2, language:=english);
nn.name:='BarcelonaCard';
assert occurrence np2;
barcelonaCard:=np2.createdProduct;

apsc1 := new AddProductToShoppingCart(quantity:=1);
apsc1.session:=ns.createdSession;
apsc1.product := barcelonaMap;
assert occurrence apsc1;

apsc2 := new AddProductToShoppingCart(quantity:=4);
```

```
apsc2.session:=ns.createdSession;
apsc2.product := barcelonaCard;
assert occurrence apsc2;

test updateShoppingCart{

       usc := new UpdateShoppingCart;
       usc.shoppingCart := ns.createdSession.shoppingCart;
       lc1 := new LineChange(updateShoppingCart:=usc,
            remove:=true, quantity:=1);
       lc2 := new LineChange(updateShoppingCart:=usc,
           remove:=false, quantity:=3);

       assert occurrence usc;
}
}
```

The test case *updateShoppingCart* asserts the occurrence of the event *UpdateShoppingCart* in an scenario where two products have been added to the anonymous shopping cart of an opened session.

Figure 2 shows the coverage results after adding the previous test case:

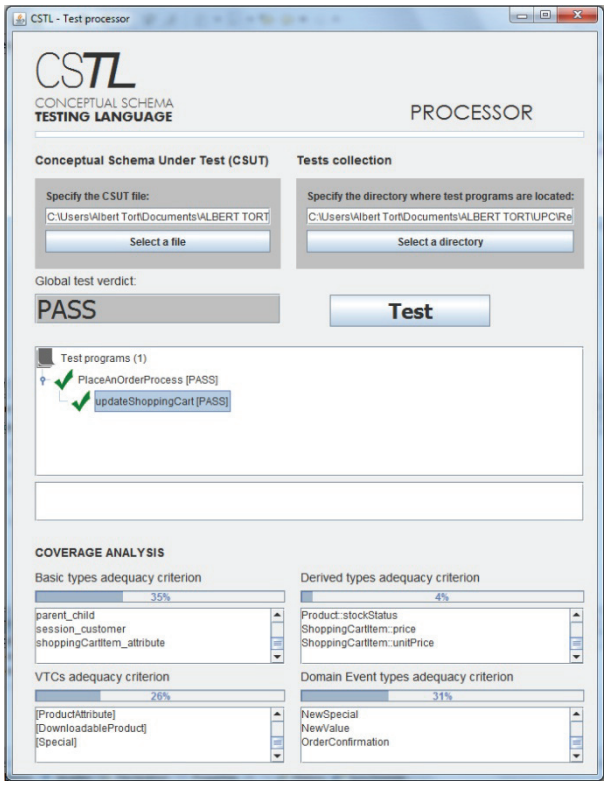


Figure 2. Coverage results (Step 1).

Uncovered elements report

==Uncovered Entity types==
Attribute
Category
Customer
CustomerShoppingCart
DownloadableProduct
LogIn
NewAttribute
NewCategory
NewCustomer
NewDownloadableProduct
NewDownloadableSpecial
NewOption
NewProductAttribute
NewSpecial
NewValue
Option
Order
OrderConfirmation
OrderLine
OrderLineAttribute

ProductAttribute
Special
Value

==Uncovered Relationship types==
Attribute
ProductAttribute
addProductToShoppingCart_attribute
category_product
customerShoppingCart_customer
customer_order
logIn_customer
logIn_session
newAttribute_option
newAttribute_value
newCategory_category
newProductAttribute_option
newProductAttribute_product
newProductAttribute_value
newProduct_category
orderConfirmation_customerShoppingC
art
orderLineAttribute_attribute
orderLine_orderLineAttribute
orderLine_product
order_orderLine
parent_child
session_customer
shoppingCartItem_attribute

==Uncovered Derived Types==
Category::products
Category::subcategories
Order::eMail
Order::id
Order::name
Order::purchased
Order::total
OrderLine::basePrice
OrderLine::name
OrderLine::price
OrderLine::unitPrice
OrderLineAttribute::increment
OrderLineAttribute::option
OrderLineAttribute::sign
OrderLineAttribute::value
Product::finalNetPrice
Product::quantityOrdered
Product::stockStatus
ShoppingCartItem::price
ShoppingCartItem::unitPrice

==Uncovered VTCs==
[DownloadableProduct, Special]
[CustomerShoppingCart]
[Customer]
[OrderLine]
[Category]
[Category]
[Order]
[OrderLineAttribute]
[Attribute]
[Option]
[Value]
[ProductAttribute]
[DownloadableProduct]
[Special]

==Uncoverent Domain Events==
LogIn
NewAttribute
NewCategory
NewCustomer
NewDownloadableProduct
NewDownloadableSpecial
NewOption
NewProductAttribute
NewSpecial
NewValue
OrderConfirmation

In the previous test program fragment we only deal with products without categories. After analyzing the set of uncovered elements, we can modify the fixture and adding a new test case in order to exercise categories. We can also add some assertions to the test case *updateShoppingCart* in order to test the derived attribute *ShoppingCartItem::price* (note that its derivation rule also exercises the derived attributes *ShoppingCartItem::unitPrice* and *Product::finalNetPrice*).

```
testprogram PlaceAnOrderProcess {

//FIXTURE
nlang := new NewLanguage(code:='en', name:='english');
assert occurrence nlang;
english:=nlang.createdLanguage;

ns := new NewSession;
assert occurrence ns;

nc1 := new NewCategory;
nn:= new NewName(namedElementEvent:=nc1, language:=english);
nn.name:='Tourism';
assert occurrence nc1;
tourism:=nc1.createdCategory;

nc2 := new NewCategory;
nn:= new NewName(namedElementEvent:=nc2, language:=english);
nn.name:='CityMaps';
nc2.parent:=tourism;
assert occurrence nc2;
cityMaps:=nc2.createdCategory;

np1 := new NewProduct(netPrice:=3, quantityOnHand:=50);
nn:= new NewName(namedElementEvent:=np1, language:=english);
nn.name:='BarcelonaMap';
np1.category:=Set{cityMaps};
assert occurrence np1;
barcelonaMap:=np1.createdProduct;


np2 := new NewProduct(netPrice:=30, quantityOnHand:=5);
nn:= new NewName(namedElementEvent:=np2, language:=english);
nn.name:='BarcelonaCard';
np2.category:=Set{tourism};
assert occurrence np2;
barcelonaCard:=np2.createdProduct;
```

```
apsc1 := new AddProductToShoppingCart(quantity:=1);
apsc1.session:=ns.createdSession;
apsc1.product := barcelonaMap;
assert occurrence apsc1;

apsc2 := new AddProductToShoppingCart(quantity:=4);
apsc2.session:=ns.createdSession;
apsc2.product := barcelonaCard;
assert occurrence apsc2;

test updateShoppingCart{

        usc := new UpdateShoppingCart;
        usc.shoppingCart := ns.createdSession.shoppingCart;
        lc1 := new LineChange(updateShoppingCart:=usc,
              remove:=true, quantity:=1);
        lc2 := new LineChange(updateShoppingCart:=usc,
             remove:=false, quantity:=3);

        assert occurrence usc;
        assert equals usc.shoppingCart.shoppingCartItem->at(1).quantity 3;
        assert equals usc.shoppingCart.shoppingCartItem->at(1).price() 90;
}

test categories{
        assert true cityMaps.product=Set{barcelonaMap};
        assert equals cityMaps.products() 1;
        assert equals cityMaps.subcategories() 0;
        assert true tourism.product=Set{barcelonaCard};
        assert equals tourism.products() 2;
        assert equals tourism.subcategories() 1;
}
}
```

Figure 3 shows the coverage results at this point:

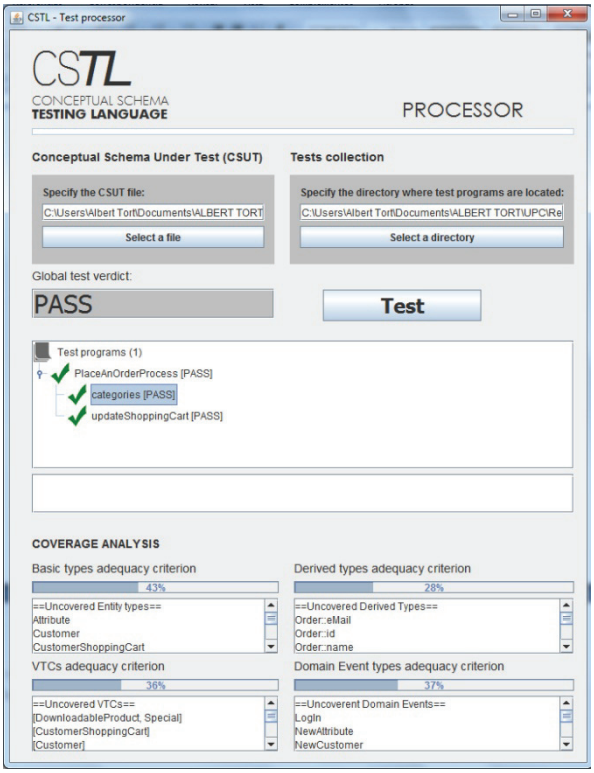


Figure 3. Coverage results (Step 2).

Uncovered elements report

==Uncovered Entity types==
Attribute
Customer
CustomerShoppingCart
DownloadableProduct
LogIn
NewAttribute
NewCustomer
NewDownloadableProduct
NewDownloadableSpecial
NewOption
NewProductAttribute
NewSpecial
NewValue
Option
Order
OrderConfirmation
OrderLine
OrderLineAttribute
ProductAttribute
Special
Value

==Uncovered Relationship types==
Attribute
ProductAttribute
addProductToShoppingCart_attribute
customerShoppingCart_customer
customer_order
logIn_customer
logIn_session
newAttribute_option
newAttribute_value
newProductAttribute_option
newProductAttribute_product
newProductAttribute_value
orderConfirmation_customerShoppingCart
orderLineAttribute_attribute
orderLine_orderLineAttribute
orderLine_product
order_orderLine
session_customer
shoppingCartItem_attribute

==Uncovered Derived Types==
Order::eMail
Order::id
Order::name
Order::purchased
Order::total
OrderLine::basePrice
OrderLine::name
OrderLine::price
OrderLine::unitPrice
OrderLineAttribute::increment
OrderLineAttribute::option
OrderLineAttribute::sign
OrderLineAttribute::value
Product::quantityOrdered
Product::stockStatus

==Uncovered VTCs==
[DownloadableProduct, Special]
[CustomerShoppingCart]
[Customer]
[OrderLine]
[Order]
[OrderLineAttribute]
[Attribute]
[Option]
[Value]
[ProductAttribute]
[DownloadableProduct]
[Special]

==Uncoverent Domain Events==
LogIn
NewAttribute
NewCustomer
NewDownloadableProduct
NewDownloadableSpecial
NewOption
NewProductAttribute
NewSpecial
NewValue
OrderConfirmation

At this point, most of the uncovered elements are related to orders and can only be tested by confirming an order. We add the test case *ConfirmOrder* in order to make progress in the fulfillment of the *basic set of test adequacy criteria.* Note that we also assert the contents of the IB after the order confirmation in order to cover the derived attributes *Product::quantityOrdererd*, *Product::stockStatus* and all the constant derived attributes of the entity types *Order* and *OrderLine*.

```
testprogram PlaceAnOrderProcess {

//FIXTURE
nlang := new NewLanguage(code:='en', name:='english');
assert occurrence nlang;
english:=nlang.createdLanguage;

ns := new NewSession;
assert occurrence ns;

nc1 := new NewCategory;
nn:= new NewName(namedElementEvent:=nc1, language:=english);
nn.name:='Tourism';
assert occurrence nc1;
tourism:=nc1.createdCategory;

nc2 := new NewCategory;
nn:= new NewName(namedElementEvent:=nc2, language:=english);
nn.name:='CityMaps';
nc2.parent:=tourism;
assert occurrence nc2;
cityMaps:=nc2.createdCategory;

np1 := new NewProduct(netPrice:=3, quantityOnHand:=50);
nn:= new NewName(namedElementEvent:=np1, language:=english);
nn.name:='BarcelonaMap';
np1.category:=Set{cityMaps};
assert occurrence np1;
barcelonaMap:=np1.createdProduct;
```

```
np2 := new NewProduct(netPrice:=30, quantityOnHand:=5);
nn:= new NewName(namedElementEvent:=np2, language:=english);
nn.name:='BarcelonaCard';
np2.category:=Set{tourism};
assert occurrence np2;
barcelonaCard:=np2.createdProduct;
apsc1 := new AddProductToShoppingCart(quantity:=1);
apsc1.session:=ns.createdSession;
apsc1.product := barcelonaMap;
assert occurrence apsc1;

apsc2 := new AddProductToShoppingCart(quantity:=4);
apsc2.session:=ns.createdSession;
apsc2.product := barcelonaCard;
assert occurrence apsc2;

test updateShoppingCart{
        usc := new UpdateShoppingCart;
        usc.shoppingCart := ns.createdSession.shoppingCart;
        lc1 := new LineChange(updateShoppingCart:=usc,
              remove:=true, quantity:=1);
        lc2 := new LineChange(updateShoppingCart:=usc,
             remove:=false, quantity:=3);

        assert occurrence usc;
}

test categories{
        assert true cityMaps.product=Set{barcelonaMap};
        assert equals cityMaps.products() 1;
        assert equals cityMaps.subcategories() 0;
        assert true tourism.product=Set{barcelonaCard};
        assert equals tourism.products() 2;
        assert equals tourism.subcategories() 1;
}

test confirmOrder{
        nc := new NewCustomer
               (firstName:='John', lastName:='James', eMailAddress:='john@james.com',
                phone:='9999999990', password:='password');
        assert occurrence nc;
        li := new LogIn;
        li.customer := nc.createdCustomer;
        li.session := ns.createdSession;
        assert occurrence li;

        oc := new OrderConfirmation;
        oc.shoppingCart := ns.createdSession.customer.customerShoppingCart;
        assert occurrence oc;

        assert equals barcelonaCard.quantityOrdered() 4;
        assert equals barcelonaMap.quantityOrdered() 1;
        assert equals barcelonaCard.stockStatus() #inStock;
        assert equals barcelonaMap.stockStatus() #inStock;

        assert equals oc.createdOrder.name() 'John James';
        assert equals oc.createdOrder.eMail() 'john@james.com';
        assert equals oc.createdOrder.purchased() '23/10/2009 20:00';
        assert equals oc.createdOrder.total() 123;
        assert equals oc.createdOrder.id() 1;
        assert equals oc.createdOrder.orderLine->at(1).name() 'BarcelonaMap';
        assert equals oc.createdOrder.orderLine->at(2).name() 'BarcelonaCard';
}

}
```

Figure 4 shows the results of the coverage analysis.

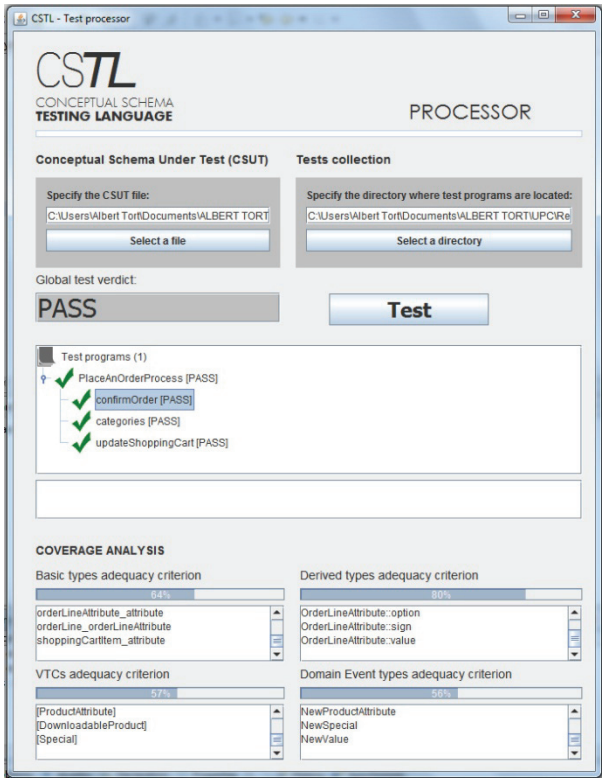


Figure 4. Coverage results (Step 3).

Uncovered elements report

==Uncovered Entity types==
Attribute
DownloadableProduct
NewAttribute
NewDownloadableProduct
NewDownloadableSpecial
NewOption
NewProductAttribute
NewSpecial
NewValue
Option
OrderLineAttribute
ProductAttribute
Special
Value

==Uncovered Relationship types==
Attribute
ProductAttribute
addProductToShoppingCart_attribute
newAttribute_option
newAttribute_value
newProductAttribute_option
newProductAttribute_product
newProductAttribute_value
orderLineAttribute_attribute
orderLine_orderLineAttribute
shoppingCartItem_attribute

==Uncovered Derived Types==
OrderLineAttribute::increment
OrderLineAttribute::option
OrderLineAttribute::sign
OrderLineAttribute::value

==Uncovered VTCs==
[DownloadableProduct, Special]
[OrderLineAttribute]
[Attribute]
[Option]
[Value]
[ProductAttribute]
[DownloadableProduct]
[Special]

==Uncoverent Domain Events==
NewAttribute
NewDownloadableProduct
NewDownloadableSpecial
NewOption
NewProductAttribute
NewSpecial
NewValue

By analyzing the set of uncovered elements we realize that although we tested the confirmation of an order, the aim of satisfying the *basic set of test adequacy criteria* forces exercising the confirmation of an order given a shopping cart consisting of product items with attributes. Consider the new test case *ConfirmOrderWithAttributes* included in the test program *PlaceAnOrderProcess.*

```
testprogram PlaceAnOrderProcess {

//FIXTURE
nlang := new NewLanguage(code:='en', name:='english');
assert occurrence nlang;
english:=nlang.createdLanguage;

ns := new NewSession;
assert occurrence ns;

nc1 := new NewCategory;
nn:= new NewName(namedElementEvent:=nc1, language:=english);
nn.name:='Tourism';
assert occurrence nc1;
tourism:=nc1.createdCategory;

nc2 := new NewCategory;
nn:= new NewName(namedElementEvent:=nc2, language:=english);
nn.name:='CityMaps';
nc2.parent:=tourism;
assert occurrence nc2;
cityMaps:=nc2.createdCategory;

np1 := new NewProduct(netPrice:=3, quantityOnHand:=50);
nn:= new NewName(namedElementEvent:=np1, language:=english);
nn.name:='BarcelonaMap';
np1.category:=Set{cityMaps};
assert occurrence np1;
barcelonaMap:=np1.createdProduct;


np2 := new NewProduct(netPrice:=30, quantityOnHand:=5);
nn:= new NewName(namedElementEvent:=np2, language:=english);
nn.name:='BarcelonaCard';
np2.category:=Set{tourism};
assert occurrence np2;
barcelonaCard:=np2.createdProduct;

apsc1 := new AddProductToShoppingCart(quantity:=1);
apsc1.session:=ns.createdSession;
apsc1.product := barcelonaMap;
assert occurrence apsc1;

apsc2 := new AddProductToShoppingCart(quantity:=4);
apsc2.session:=ns.createdSession;
apsc2.product := barcelonaCard;
assert occurrence apsc2;

test updateShoppingCart{

        usc := new UpdateShoppingCart;
        usc.shoppingCart := ns.createdSession.shoppingCart;
        lc1 := new LineChange(updateShoppingCart:=usc,
              remove:=true, quantity:=1);
        lc2 := new LineChange(updateShoppingCart:=usc,
             remove:=false, quantity:=3);

        assert occurrence usc;
        assert equals usc.shoppingCart.shoppingCartItem->at(1).quantity 3;
        assert equals usc.shoppingCart.shoppingCartItem->at(1).price() 90;
}

test categories{
        assert true cityMaps.product=Set{barcelonaMap};
        assert equals cityMaps.products() 1;
        assert equals cityMaps.subcategories() 0;
        assert true tourism.product=Set{barcelonaCard};
        assert equals tourism.products() 2;
        assert equals tourism.subcategories() 1;
}
```

```
test confirmOrder{
        nc := new NewCustomer
               (firstName:='John', lastName:='James', eMailAddress:='john@james.com',
                phone:='9999999990', password:='password');
        assert occurrence nc;
        li := new LogIn;
        li.customer := nc.createdCustomer;
        li.session := ns.createdSession;
        assert occurrence li;

        oc := new OrderConfirmation;
        oc.shoppingCart := ns.createdSession.customer.customerShoppingCart;
        assert occurrence oc;

        assert equals barcelonaCard.quantityOrdered() 4;
        assert equals barcelonaMap.quantityOrdered() 1;
        assert equals barcelonaCard.stockStatus() #inStock;
        assert equals barcelonaMap.stockStatus() #inStock;
}

test confirmOrderWithAttributes{
        no1 := new NewOption;
        nn:= new NewName(namedElementEvent:=no1, language:=english);
        nn.name:='Age';
        assert occurrence no1;
        age := no1.createdOption;

        nv1 := new NewValue;
        nn:= new NewName(namedElementEvent:=nv1, language:=english);
        nn.name:='Child';
        assert occurrence nv1;
        child := nv1.createdValue;

        na := new NewAttribute;
        na.option:=age;
        na.value:=child;
        assert occurrence na;
        childAge:=na.createdAttribute;

        npa := new NewProductAttribute(sign:=#minus, increment:=8);
        npa.product:=barcelonaCard;
        npa.option:=age;
        npa.value:=child;
        assert occurrence npa;

        apsc3 := new AddProductToShoppingCart(quantity:=1);
        apsc3.session:=ns.createdSession;
        apsc3.product := barcelonaCard;
        apsc3.attribute:=Set{childAge};
        assert occurrence apsc3;

        nc := new NewCustomer
               (firstName:='Mary', lastName:='Johnes', eMailAddress:='mary@johnes.com',
                phone:='9999999990', password:='password');
        assert occurrence nc;
        li := new LogIn;
        li.customer := nc.createdCustomer;
        li.session := ns.createdSession;
        assert occurrence li;

        oc := new OrderConfirmation;
        oc.shoppingCart := ns.createdSession.customer.customerShoppingCart;
        assert occurrence oc;

        assert equals barcelonaCard.quantityOrdered() 5;
        assert equals barcelonaMap.quantityOrdered() 1;
        assert equals barcelonaCard.stockStatus() #outOfStock;
        assert equals barcelonaMap.stockStatus() #inStock;

        assert equals oc.createdOrder.name() 'Mary Johnes';
        assert equals oc.createdOrder.eMail() 'mary@johnes.com';
        assert equals oc.createdOrder.purchased() '23/10/2009 20:00';
```

```
        assert equals oc.createdOrder.total() 145;
        assert equals oc.createdOrder.id() 1;
        assert equals oc.createdOrder.orderLine->at(1).name() 'BarcelonaMap';
        assert equals oc.createdOrder.orderLine->at(2).name() 'BarcelonaCard';
        assert equals oc.createdOrder.orderLine->at(3).name() 'BarcelonaCard';
        assert equals oc.createdOrder.orderLine->at(3).orderLineAttribute.option()
                       ->any(true) 'Age';
        assert equals oc.createdOrder.orderLine->at(3).orderLineAttribute.value()
                       ->any(true) 'Child';
}
}
```

Figure 5 shows the coverage analysis results after adding the test case *confirmOrderWithAttributes*:

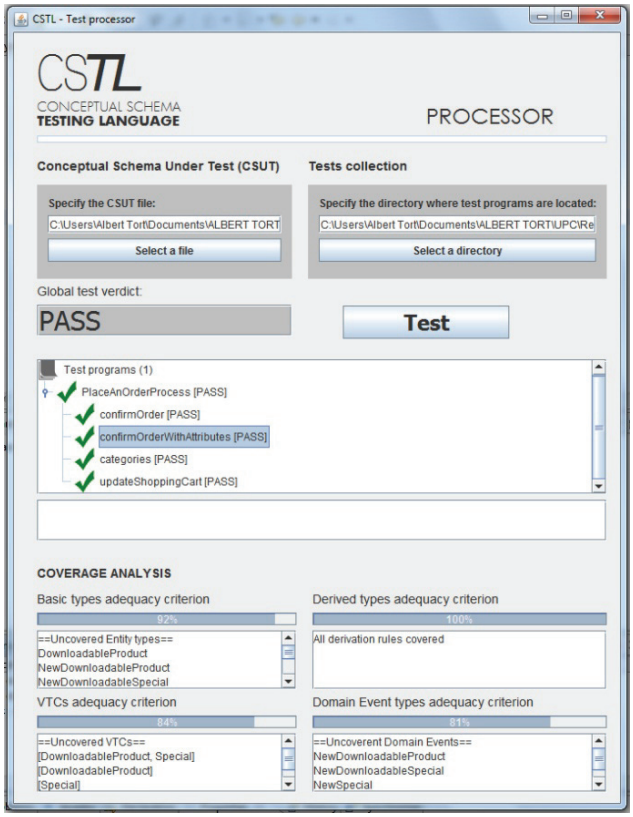


Figure 5. Coverage results (Step 4).

Uncovered elements report

==Uncovered Entity types==
DownloadableProduct
NewDownloadableProduct
NewDownloadableSpecial
NewSpecial
Special

==Uncovered VTCs==
[DownloadableProduct, Special]
[DownloadableProduct]
[Special]

==Uncoverent Domain Events==
NewDownloadableProduct
NewDownloadableSpecial
NewSpecial

Finally, in order to achieve a test set that satisfies at all the *basic set of test adequacy criteria,* we consider the new test program *ProductKinds* in order to exercise the different kinds of products:

```
testprogram productKinds{

nlang := new NewLanguage(code:='en', name:='english');
assert occurrence nlang;
english:=nlang.createdLanguage;

test specials{
       s1 := new NewSpecial(netPrice:=5, quantityOnHand:=30, specialNetPrice:=4);
       nn:= new NewName(namedElementEvent:=s1, language:=english);
       nn.name:='AudioGuide';
       assert occurrence s1;
}

test downloadableProducts{
       dp1 := new NewDownloadableProduct(netPrice:=23, quantityOnHand:=200,
                  downloadLink:='http://touristdownloads/barcelona.exe');
       nn:= new NewName(namedElementEvent:=dp1, language:=english);
       nn.name:='BarcelonaVirtualMap';
       assert occurrence dp1;
}

test downloadableSpecial{
       ds1 := new NewDownloadableSpecial(netPrice:=23, quantityOnHand:=200,
            downloadLink:='http://touristdownloads/barcelona.exe', specialNetPrice:=20);
       nn:= new NewName(namedElementEvent:=ds1, language:=english);
       nn.name:='BarcelonaVirtualMap';
       assert occurrence ds1;
}
}
```

Figure 6 shows the coverage results given by the CSTL test processor.

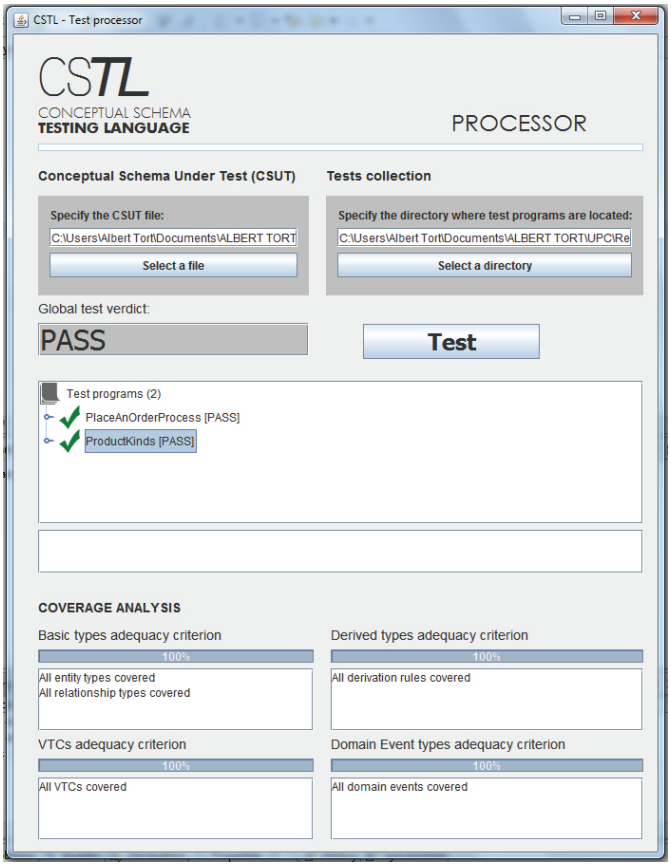


Figure 6. Coverage results (Step 5).

The main conclusion of these results is that the set of test cases specified in the test programs *PlaceAnOrderProcess* and *ProductKinds* are adequate according to the *basic set of test adequacy criteria* in order to ensure the satisfiability of all the elements of the fragment of the conceptual schema presented in Section 2.

References

1. Olivé, A. Conceptual Modeling of Information Systems. Springer, 2007.
2. Olivé, A; Raventós, R. "Modeling events as entities in object-oriented conceptual modeling languages". Data&Knowledge Engineering 58 (2006) pp. 243-262.
3. OMG. Object Constraint Language (OCL). Version 2.0 May 2006.
4. OMG. UML Supraestructure version 2.1.2, November 2007.
5. Tort, A. The osCommerce Conceptual Schema. http://guifre.lsi.upc.edu/OSCommerce.pdf , 2007.
6. Tort, A. Testing the osCommerce Conceptual Schema by Using CSTL, Research Report LSI-09-28-R, UPC, 2009. Available from: <http://www.lsi.upc.edu/~techreps/files/R09-28.zip>
7. Tort, A; Olivé, A. First Steps Towards Conceptual Schema Testing, in: Proceeding of CAiSE Forum 2009. Available from: <http://ceur-ws.org/Vol-453>
8. Tort, A; Olivé, A. An Approach to Testing Conceptual Schemas. 2009. *Submitted for publication.*

Annex 1: The fragment of the osCommerce conceptual schema in the USE notation

```
model basicOSCommerce

--Enumerations
enum ProductStatus{inStock,outOfStock}
enum Sign{plus,minus}
enum Status{enabled,disabled}
enum OrderStatus{pending,delivered,cancelled}

--System
class System
operations
        Now():String='23/10/2009 10:00'
end

--Entity types

class ShoppingCart
end

class AnonymousShoppingCart < ShoppingCart
end

class CustomerShoppingCart < ShoppingCart
end

class NamedElement
end

class Customer
attributes
        firstName:String
        lastName:String
        eMailAddress:String
        phone:String
        password:String
        numberOfLogons:Integer
        status:Status
end

class Session
attributes
        _sessionID: Integer
        ipAddress:String
        timeEntry:String
operations
        sessionID():Integer=Session.allInstances->size()
end

class Order
attributes
        _id:Integer
        _name:String
        _eMail:String
        _purchased:String
        _total:Real
        currentStatus:OrderStatus
operations
        id():Integer=
                if Order.allInstances->size()=1 then 1
                else (Order.allInstances->excluding(self))->sortedBy(id()) -> last().id()+1
                endif
        name():String=
                self.customer.firstName
                .concat(' ')
                .concat(self.customer.lastName)
        eMail():String=self.customer.eMailAddress
        purchased():String='23/10/2009 20:00'
        total():Real=self.orderLine.price()->sum()

end
```

```
class OrderLine
attributes
        _name:String
        _basePrice:Real
        _unitPrice:Real
        _price:Real
        quantity:Integer
operations
        name():String=self.product.nameInLanguage->any(true).name
        basePrice():Real=self.product.netPrice
        unitPrice():Real=
                self.orderLineAttribute
                        ->collect
                        (if sign()=#plus then increment()
                         else -increment()
                         endif)->sum() + self.basePrice()
        price():Real=self.unitPrice()*self.quantity
end

class OrderLineAttribute
attributes
        _option:String
        _value:String
        _increment:Real
        _sign:Sign
operations
        option():String=
                self.attribute.option.nameInLanguage->any(true).name
        value():String=
                self.attribute.value.nameInLanguage->any(true).name
        increment():Real=
                self.attribute.productAttribute
                        -> select(pa | pa.product=self.orderLine.product)
                        ->any(true).increment
        sign():Sign=
                self.attribute.productAttribute
                        -> select(pa | pa.product=self.orderLine.product)
                        ->any(true).sign
end


class Category < NamedElement
attributes
        _subcategories:Integer
        _products:Integer
operations
        allParents():Set(Category)=if self.parent->notEmpty then self.parent
         ->union(self.parent.allParents()) else oclEmpty(Set(Category)) endif
        subcategories():Integer=Category.allInstances -> select(c| c.allParents()-> includes(self))
         ->size()
        products():Integer=
        Category.allInstances -> select(c|
        c.allParents()-> includes(self) or        c=self).product->size()
end

class Product < NamedElement
attributes
        _stockStatus:ProductStatus
        status:Status
        netPrice:Real
        _finalNetPrice:Real
        quantityOnHand:Integer
        _quantityOrdered:Integer
operations
        quantityOrdered():Integer=self.orderLine.quantity->sum()
        finalNetPrice():Real=if self.oclIsTypeOf(Special) then
           self.oclAsType(Special).specialNetPrice else netPrice endif
        stockStatus():ProductStatus=if quantityOnHand>0 then #inStock else #outOfStock endif
end

class Special < Product
attributes
        specialNetPrice:Real
end

class DownloadableProduct < Product
attributes
        downloadLink:String
end

class Option < NamedElement
end
```

```
class Value < NamedElement
end

class ShoppingCartItem
attributes
        quantity:Integer
        _unitPrice:Real
        _price:Real
operations
        unitPrice():Real=
                self.attribute.productAttribute
                        ->select(pa | pa.product=self.product)
                        ->collect
                        (if sign=#plus then increment
                         else -increment
                         endif)->sum() + self.product.finalNetPrice()
        price():Real=self.unitPrice()*self.quantity
end

class Language
attributes
        name:String
        code:String
        status:Status
end

--Association classes

associationclass Attribute between
        Option[*]
        Value[*]
end

associationclass ProductAttribute between
        Product[*]
        Attribute[*]
attributes
        increment:Real
        sign:Sign
        status:Status
end

associationclass NameInLanguage between
        NamedElement[*]
        Language[*]
attributes
        name:String
end


-- Associations

association customerShoppingCart_customer between
        CustomerShoppingCart[0..1]
        Customer[1]
end

association shoppingCart_session between
        ShoppingCart[0..1]
        Session[0..1]
end

association anonymousShoppingCart_session between
        AnonymousShoppingCart[0..1]
        Session[0..1] role redefinedSession
end

association session_customer between
        Session[0..1]
        Customer[0..1]
end

association customer_order between
        Customer[1]
        Order[*]
end

association order_orderLine between
        Order[1]
        OrderLine[1..*] ordered
end
```

```
association orderLine_orderLineAttribute between
        OrderLine[1]
        OrderLineAttribute[*]
end

association orderLineAttribute_attribute between
        OrderLineAttribute[*]
        Attribute[1]
end


association parent_child between
        Category[0..1] role parent
        Category[*] role child
end

association category_product between
        Category[*]
        Product[*]
end

association orderLine_product between
        OrderLine[*]
        Product[1]
end

association shoppingCart_shoppingCartItem between
        ShoppingCart[1]
        ShoppingCartItem[1..*] ordered
end

association shoppingCartItem_product between
        ShoppingCartItem[*]
        Product[1]
end

association shoppingCartItem_attribute between
        ShoppingCartItem[*]
        Attribute[*]
end

-- BEHAVIORAL SCHEMA

abstract class Event
operations
  effect()
end

abstract class DomainEvent < Event
end

class UpdateShoppingCart < DomainEvent
operations
        effect()
end

class LineChange
attributes
        remove:Boolean
        quantity:Integer
end

association updateShoppingCart_lineChange between
        UpdateShoppingCart[*]
        LineChange[1..*] ordered
end

association updateShoppingCart_shoppingCart between
        UpdateShoppingCart[*]
        ShoppingCart[1]
end

class OrderConfirmation < DomainEvent
attributes
    createdOrder:Order
operations
    effect()
end

association orderConfirmation_customerShoppingCart between
    OrderConfirmation[*]
    CustomerShoppingCart[0..1] role shoppingCart
end
```

```
class NewCustomer < DomainEvent
attributes
        firstName:String
        lastName:String
        eMailAddress:String
        phone:String
        password:String
        createdCustomer:Customer
operations
    effect()
end

abstract class NamedElementEvent < DomainEvent
end

associationclass NewName between
    NamedElementEvent[*]
    Language[*]
attributes
        name:String
end

class NewCategory < NamedElementEvent
attributes
        createdCategory:Category
operations
    effect()
end

association newCategory_category between
    NewCategory[*]
    Category[0..1] role parent
end

class NewSession < DomainEvent
attributes
        createdSession:Session
operations
    effect()
end

class NewOption < NamedElementEvent
attributes
        createdOption:Option
operations
    effect()
end

class NewValue < NamedElementEvent
attributes
        createdValue:Value
operations
    effect()
end

class NewAttribute
attributes
        createdAttribute:Attribute
operations
    effect()
end


association newAttribute_option between
    NewAttribute[*]
    Option[1]
end

association newAttribute_value between
    NewAttribute[*]
    Value[1]
end

class NewProductAttribute < DomainEvent
attributes
    increment:Real
    sign:Sign
        createdProductAttribute:ProductAttribute
operations
    effect()
end
```

```
association newProductAttribute_option between
    NewProductAttribute[*]
    Option[1]
end

association newProductAttribute_value between
    NewProductAttribute[*]
    Value[1]
end

association newProductAttribute_product between
    NewProductAttribute[*]
    Product[1]
end


class AddProductToShoppingCart < DomainEvent
attributes
    quantity:Integer
operations
    effect()
end

association addProductToShoppingCart_attribute between
    AddProductToShoppingCart[*]
    Attribute[*]
end

association addProductToShoppingCart_product between
    AddProductToShoppingCart[*]
    Product[1]
end

association addProductToShoppingCart_session between
    AddProductToShoppingCart[*]
    Session[1]
end

class NewProduct < NamedElementEvent
attributes
    netPrice:Real
    quantityOnHand:Integer
        createdProduct:Product
operations
    effect()
end

association newProduct_category between
    NewProduct[*]
    Category[*]
end

class NewDownloadableProduct < NewProduct
attributes
    downloadLink:String
        createdDownloadableProduct:DownloadableProduct
operations
    effect()
end

class NewSpecial < NewProduct
attributes
    specialNetPrice:Real
        createdSpecial:Special
operations
    effect()
end


class NewDownloadableSpecial < NewSpecial,NewDownloadableProduct
operations
        effect()
end

class LogIn < DomainEvent
operations
    effect()
end

association logIn_session between
    LogIn[*]
    Session[1]
end
```

```
association logIn_customer between
    LogIn[*]
    Customer[1]
end

class NewLanguage < DomainEvent
attributes
    code:String
        name:String
        createdLanguage:Language
operations
    effect()
end


-- OCL constraints

constraints

context Language
  inv codeAndNameAreUnique:
        Language.allInstances -> isUnique(name) and
        Language.allInstances ->isUnique(code)

context NamedElement
  inv nameIsUnique:
        self.language->forAll(nameInLanguage->isUnique(name))

 context NamedElement
   inv aNameInEachLanguage:
         self.language=Language.allInstances

 context Category
    inv isAHierarchy:
           not self.allParents()->includes(self)

context Customer
  inv eMailIsUnique:
        Customer.allInstances->isUnique(eMailAddress)

context Session
  inv sessionIDIsUnique:
        Session.allInstances->isUnique(sessionID())

context CustomerShoppingCart
  inv sameCustomer:
        self.session.customer->notEmpty() implies
        self.session.customer=self.customer

context ShoppingCartItem
  inv productHasTheAttributes:
        self.product.attribute->includesAll(self.attribute)

context ShoppingCartItem
  inv onlyOneAttributePerOption:
        self.attribute->isUnique(option)

context Order
  inv idIsUnique:
        Order.allInstances->isUnique(id())

-- Event constraints
context UpdateShoppingCart
  inv _iniIC_complete:
        self.lineChange->size() =
        self.shoppingCart.shoppingCartItem->size()

context NewCustomer inv _iniIC_customerDoesNotExist:
     not Customer.allInstances -> exists (c | c.eMailAddress=self.eMailAddress)

context NamedElementEvent inv _iniIC_aNameInEachLanguage:
        Language.allInstances = self.language

context NewCategory inv _iniIC_categoryDoesNotExist:
        not Category.allInstances->exists(c | c.nameInLanguage.name = self.newName.name)

context NewOption inv _iniIC_optionDoesNotExist:
        not Option.allInstances->exists(o | o.nameInLanguage.name = self.newName.name)

context NewLanguage inv _iniIC_languageDoesNotExist:
        not Language.allInstances->exists(l | l.code=self.code or l.name=self.name)
```

```
context NewValue inv _iniIC_valueDoesNotExist:
        not Value.allInstances->exists(v | v.nameInLanguage.name = self.newName.name)

context NewProductAttribute inv _iniIC_productAttributeDoesNotExist:
      not self.product.productAttribute ->
                 exists(attribute.value=self.value and
                     attribute.option = self.option)

context NewProductAttribute inv _iniIC_optionValueIsValid:
    self.option.value -> includes(self.value)

context NewAttribute inv _iniIC_attributeDoesNotExist:
      not Attribute.allInstances -> exists(a | a.value = self.value and a.option = self.option)

context AddProductToShoppingCart inv _iniIC_AttributesAreFromProduct:
  self.product.attribute -> includesAll(self.attribute)

context AddProductToShoppingCart inv _iniIC_AttributesAreOfDifferentOptions:
  self.attribute -> isUnique(option)

context NewProduct inv _iniIC_productDoesNotExist:
        not Product.allInstances->exists(p | p.nameInLanguage.name = self.newName.name)

context LogIn inv _iniIC_CustomerIsNotLoggedIn:
   self.customer.session -> isEmpty()

-- EFFECT OPERATIONS
context  UpdateShoppingCart::effect()
   post UpdateShoppingCartEffect:
       self.lineChange ->forAll
          (lc|let cartItem:ShoppingCartItem =
                    let i:Integer=Set{1,2,3,4,5}->select(i | self.lineChange->at(i)=lc)->any(true)
                                        in
                                        self.shoppingCart.shoppingCartItem@pre->at(i)
               in
                (lc.remove or lc.quantity <> cartItem.quantity)
                   implies
                     if lc.remove then
                       (ShoppingCartItem.allInstances)
                       -> excludes(cartItem)
                     else
                       cartItem.quantity = lc.quantity
                     endif )

context  OrderConfirmation::effect()
   post theOrderIsCreated:
   (Order.allInstances - Order.allInstances@pre) -> one(o:Order |
       o.oclIsNew() and
       o.oclIsTypeOf(Order) and
       self.createdOrder=o and
       o.customer = self.shoppingCart@pre.customer@pre and
       --The initial status of the order
       o.currentStatus = #pending and
       --There is an order line for each shopping cart item
      shoppingCart@pre.shoppingCartItem@pre->forAll
         (i|OrderLine.allInstances -> one
            (ol|ol.order = o  and
                  ol.product = i.product@pre  and
                  ol.quantity = i.quantity@pre  and
                  i.attribute@pre->forAll
                      (iAtt|OrderLineAttribute.allInstances -> exists
                      (olAtt|olAtt.orderLine = ol and
                                 olAtt.attribute = iAtt)))))
   post theShoppingCartIsRemoved:
       ShoppingCart.allInstances->excludes(self.shoppingCart@pre)
   post updateProductQuantities:
       let productsBought:Set(Product) =
           self.shoppingCart@pre.shoppingCartItem@pre.product@pre->asSet()
       in  productsBought -> forAll (p|
                let quantityBought:Integer =
                       self.shoppingCart@pre.shoppingCartItem@pre->select
                          (sc | sc.product = p).quantity -> sum()
                in
                       p.quantityOnHand = p.quantityOnHand@pre - quantityBought)


context  NewCustomer::effect()
    post newCustomerEffect:
   (Customer.allInstances - Customer.allInstances@pre) -> one(c:Customer |
       c.oclIsNew() and
       c.oclIsTypeOf(Customer) and
       c.firstName = self.firstName and
       c.lastName = self.lastName and
```

```
       c.eMailAddress = self.eMailAddress and
       c.phone = self.phone and
       c.password = self.password and
       c.numberOfLogons = 0 and
          c.status =#enabled and
                self.createdCustomer=c)

context  NewCategory::effect()
   post newCategoryEffect:
   (Category.allInstances - Category.allInstances@pre) -> one(c:Category |
       c.oclIsNew()  and
       c.oclIsTypeOf(Category) and
            c.parent = self.parent and
       self.newName.name=c.nameInLanguage.name and
                self.createdCategory=c)

context  NewSession::effect()
   post newSessionEffect:
    (Session.allInstances - Session.allInstances@pre) -> one(s:Session |
       s.oclIsNew() and
       s.oclIsTypeOf(Session) and
            s.ipAddress='xxx.xxx.xxx.xxx' and
            s.timeEntry='xx/xx/xxxx' and
            self.createdSession=s
       )

context  NewOption::effect()
   post newOptionEffect:
   (Option.allInstances - Option.allInstances@pre) -> one(o:Option |
       o.oclIsNew()  and
       o.oclIsTypeOf(Option) and
       self.newName.name=o.nameInLanguage.name and
            self.createdOption=o)

context  NewValue::effect()
   post newValueEffect:
   (Value.allInstances - Value.allInstances@pre) -> one(v:Value |
       v.oclIsNew()  and
       v.oclIsTypeOf(Value) and
       self.newName.name=v.nameInLanguage.name and
            self.createdValue=v)


context  AddProductToShoppingCart::effect()
   post addProductToShoppingCartEffect :
       (ShoppingCartItem.allInstances - ShoppingCartItem.allInstances@pre) ->
one(sci:ShoppingCartItem |
       sci.oclIsNew and
       sci.oclIsTypeOf(ShoppingCartItem) and
       sci.quantity = self.quantity and
       sci.product = self.product and
       sci.attribute = self.attribute and
       if self.session.shoppingCart -> notEmpty() then
           --The session has a shopping cart
           self.session.shoppingCart.shoppingCartItem -> includes(sci)
       else
           --The session does not have a shopping cart
           if self.session.customer -> isEmpty() then
           --The session is Anonymous
                (AnonymousShoppingCart.allInstances - AnonymousShoppingCart.allInstances@pre)
               -> one(sc:AnonymousShoppingCart |
               sc.oclIsNew()  and
               sc.oclIsTypeOf(AnonymousShoppingCart)  and
               self.session.shoppingCart = sc and
                                  self.session.anonymousShoppingCart = sc and
               sc.shoppingCartItem -> includes(sci))
          else
           --The customer is logged in
               if self.session.customer.customerShoppingCart -> notEmpty()  then
                   --The customer has a previous shopping cart
                  self.session.shoppingCart = self.session.customer.customerShoppingCart and
                  self.session.shoppingCart.shoppingCartItem -> includes(sci)
               else
                   --The customer does not have a previous shopping cart
                   (CustomerShoppingCart.allInstances - CustomerShoppingCart.allInstances@pre)
                   -> one(csc:CustomerShoppingCart |
                   csc.oclIsNew()  and
                   csc.oclIsTypeOf(CustomerShoppingCart)  and
                   self.session.shoppingCart = csc and
                   csc.shoppingCartItem -> includes(sci))
              endif
          endif
       endif)
```

```
context  NewProduct::effect()
   post newProductEffect:
   (Product.allInstances - Product.allInstances@pre) -> one(p:Product |
       p.oclIsNew()  and
       p.oclIsTypeOf(Product) and
       p.netPrice = self.netPrice and
       p.quantityOnHand = self.quantityOnHand and
       p.status = #enabled and
            p.category = self.category and
       self.newName.name = p.nameInLanguage.name and
            self.createdProduct=p)

context  NewSpecial::effect()
        post newProductEffect:
   (Product.allInstances - Product.allInstances@pre) -> one(p:Product |
       p.oclIsNew()  and
       p.oclIsKindOf(Product) and
       p.netPrice = self.netPrice and
       p.quantityOnHand = self.quantityOnHand and
       p.status = #enabled and
            p.category = self.category and
       self.newName.name = p.nameInLanguage.name and
            self.createdProduct=p)
   post newSpecialEffect:
   (Special.allInstances - Special.allInstances@pre) -> one(s:Special |
       s.oclIsNew()  and
       s.oclIsTypeOf(Special) and
            s.specialNetPrice = self.specialNetPrice and
            self.createdSpecial=s)

context  NewDownloadableProduct::effect()
        post newProductEffect:
   (Product.allInstances - Product.allInstances@pre) -> one(p:Product |
       p.oclIsNew()  and
       p.oclIsKindOf(Product) and
       p.netPrice = self.netPrice and
       p.quantityOnHand = self.quantityOnHand and
       p.status = #enabled and
            p.category = self.category and
       self.newName.name = p.nameInLanguage.name and
            self.createdProduct=p)
   post newDownloadableProductEffect:
   (DownloadableProduct.allInstances - DownloadableProduct.allInstances@pre)
       -> one(dp:DownloadableProduct |
       dp.oclIsNew()  and
       dp.oclIsTypeOf(DownloadableProduct) and
            dp.downloadLink = self.downloadLink and
            self.createdDownloadableProduct=dp)

context  NewDownloadableSpecial::effect()
        post newProductEffect:
   (Product.allInstances - Product.allInstances@pre) -> one(p:Product |
       p.oclIsNew()  and
       p.oclIsKindOf(Product) and
       p.netPrice = self.netPrice and
       p.quantityOnHand = self.quantityOnHand and
       p.status = #enabled and
            p.category = self.category and
       self.newName.name = p.nameInLanguage.name and
            self.createdProduct=p)
        post newSpecialEffect:
   (Special.allInstances - Special.allInstances@pre) -> one(s:Special |
       s.oclIsNew()  and
       s.oclIsKindOf(Special) and
            s.specialNetPrice = self.specialNetPrice and
            self.createdSpecial=s)
        post newDownloadableProductEffect:
   (DownloadableProduct.allInstances - DownloadableProduct.allInstances@pre)
       -> one(dp:DownloadableProduct |
       dp.oclIsNew()  and
       dp.oclIsKindOf(DownloadableProduct) and
            dp.downloadLink = self.downloadLink and
            self.createdDownloadableProduct=dp)

context  NewLanguage::effect()
   post newLanguageEffect:
   (Language.allInstances - Language.allInstances@pre) -> one(l:Language |
       l.oclIsNew()  and
       l.oclIsTypeOf(Language) and
       l.code = self.code and
            l.name = self.name and
            l.status = #enabled and
            self.createdLanguage=l)
```

```
context  NewAttribute::effect()
   post :
   (Attribute.allInstances - Attribute.allInstances@pre) -> one(a:Attribute |
          a.oclIsNew()  and
      a.oclIsTypeOf(Attribute) and
      a.option = self.option and
      a.value = self.value and
          self.createdAttribute=a)

context  NewProductAttribute::effect()
   post :
   (ProductAttribute.allInstances - ProductAttribute.allInstances@pre) -> one(pa:ProductAttribute |
              pa.oclIsNew()  and
      pa.oclIsTypeOf(ProductAttribute) and
      pa.increment = self.increment and
      pa.sign = self.sign and
      pa.product = self.product and
      pa.attribute.option = self.option and
      pa.attribute.value = self.value and
          pa.status = #enabled and
          self.createdProductAttribute=pa)

context  LogIn::effect()
   post logInIdentifySession:
       self.session.customer = self.customer
   post logInUpdateNumberOfLogons:
       self.customer.numberOfLogons = self.customer.numberOfLogons@pre + 1
   post logInRestorePreviousShoppingCart:
                let previousShoppingCart:CustomerShoppingCart = self.customer.customerShoppingCart
                in
                self.customer.customerShoppingCart->notEmpty() implies
                        (self.session.shoppingCart=previousShoppingCart and
                    previousShoppingCart.shoppingCartItem -
>includesAll(self.session.shoppingCart.shoppingCartItem) and
                        previousShoppingCart.customer=self.customer and
                        self.session.shoppingCart=previousShoppingCart)
        post logInAddAnonymousItems:
                let anonymousShoppingCart:AnonymousShoppingCart = self.session.anonymousShoppingCart
                in
                self.session.anonymousShoppingCart->notEmpty() implies
                        (let currentCustomerCart:ShoppingCart = self.session.shoppingCart
                        in
                        self.session.shoppingCart->notEmpty() and
                        currentCustomerCart.oclIsTypeOf(CustomerShoppingCart) and
                        currentCustomerCart.oclAsType(CustomerShoppingCart).customer=self.customer
                        and
                        currentCustomerCart.shoppingCartItem
                         ->includesAll(anonymousShoppingCart.shoppingCartItem))
```