Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Màster en Computació

Tesi de Màster

CSTL:
A Conceptual Schema Testing Language

Estudiant: Albert **Tort Pugibet**
Director: Antoni **Olivé Ramon**

Data: 19 de gener de 2009

Departament de Llenguatges i Sistemes Informàtics
UNIVERSITAT POLITÈCNICA DE CATALUNYA
Master in Computing

Title of thesis	**CSTL:A Conceptual Schema Testing Language**
Author	**Albert Tort Pugibet**
Advisor	**Antoni Olivé Ramon**
Date	**January 15th, 2008**
Abstract	Like any software artifact, conceptual schemas of information systems can be tested. Testing conceptual schemas has some similarities with testing programs, but there are important differences. We present a list of six kinds of tests that can be applied to conceptual schemas. Some of them require schemas comprising both the structural and the behavioral parts, but we show that it is useful to test incomplete schema fragments, even if they consist of only a few entity and relationship types, integrity constraints and derivation rules. We present CSTL, a language for writing automated tests of executable schemas written in UML/OCL. CSTL follows the style of the modern xUnit testing frameworks. Tests written in CSTL can be executed as many times as needed. We describe an implementation of a test processor, which includes a test manager and a test interpreter that coordinates the execution of the tests. Finally, we apply CSTL to the conceptual schema of a real-world information system.
Keywords	**Conceptual modeling, testing, UML, OCL**
Language	**English**
Modality	**Research work**

Acknowledgements

Thanks to my master thesis advisor Dr.Antoni Olivé for his support and guidance. I really thank him for giving me the opportunity of learning from him.

Thanks to my collegues in the *Grup de Recerca en Modelització Conceptual* for their interesting and useful comments on my research work and for their friendship.

Thanks to my family for being always close to me.

This research has partly been supported by the *Universitat Politècnica de Catalunya* (UPC) and the spanish *Ministerio de Ciencia e Innovación* and FEDER under project TIN2008-00444/TIN, Grupo Consolidado.

Table of contents

1. INTRODUCTION

Testing increases confidence in quality.

In several scientific and industrial contexts, such as medical research, civil engineering or aeronautics, testing is, clearly, a critical activity. Trying and analyzing the resultant effects of applying our solutions in concrete situations is the most used mechanism to increase our confidence about the quality of products developed by humans.

Over the last decades, software has become an intrinsic part of business and society. This is the reason because software quality has become also critical. In the United States, the Department of Commerce's National Institute of Standards and Technology reported in 2002 that software errors cost the U.S. economy an estimated $59.5 billion annually [39]. The title of that study was “The Economic Impacts of Inadequate Infrastructure for Software Testing”.

Nowadays, in information systems engineering, the need and the importance of software testing is undisputed [13]. We adopt here the precise and concise definition of testing proposed by Meyer: “To test a program is to try to make it fail”, from which the goal of testing becomes “to uncover faults by triggering failures” [23]. Many other verification techniques are used or are in research and development, but, in professional practice, testing continues to be the dominant technique.

Currently, most work in conceptual modeling assumes that conceptual schemas are executable, and therefore they are software [16,22,27]. Then, some questions naturally arise: Can we test conceptual schemas? How can we do this?

In this master thesis, we try to explore what does it mean to test conceptual schemas, and we present a language for writing automated tests of conceptual schemas.

1.1. Master Thesis Purpose

The proposal presented in this master thesis is based on the idea that as any other software artifact, conceptual schemas can be tested. Testing conceptual schemas has some similarities with testing programs, but there are important differences. In this context, we pretend to make the following main contributions to the information systems research field:

- **A catalog of test kinds** that can be applied to conceptual schemas.
- **A language** for writing automated tests of executable schemas written in UML/OCL. We named it Conceptual Schema Testing Language (CSTL).

We also present **the application of CSTL to the conceptual schema of a real-world information** system. The results, conclusions and experience acquired as a result of this application are the base for proposing improvements and future work directions.

1.2. Document Structure

The structure of the master thesis report is as follows. In the next section, we explain the research approach used to achieve the main goals of this work. After that, in Section 3, we briefly review the main concepts and the notion used to define the conceptual schemas under test. In Section 4, we introduce the main ideas about testing conceptual schemas. In Section 5 we give an overview of the related work. We present a list of six kinds of tests that can be applied to conceptual schemas in Section 6. Section 7 presents in detail the second main contribution of this master thesis: the CSTL language. Section 8 describes the testing environment and our implementation of an interpreter of CSTL programs. Some test program examples are illustrated by its application to the conceptual schema of the osCommerce [41], a popular e-commerce system, in Section 9. Finally, Section 10 summarizes the conclusions and suggests further work.

The following electronic documentation and files can be found in the master tesis' website (www.lsi.upc.edu/~atort/cstl): The complete master tesis report; the appendixs of this report; the source code and the executable jar file of the prototypical CSTL processor used to try the execution of the presented test programs; the set CSTL files containing the test program examples used in this document.

2. THE RESEARCH APPROACH

The work presented in this master thesis has been structured and guided following the main ideas of the general methodology of Design research in Information Systems [2]. According to the Association for Information Systems (AIS) *"design research involves the analysis of the use and performance of designed artifacts to understand and explain and very frequently to improve on the behavior of aspects of Information Systems. Design research is also called Improvement Research, emphasizing the problem-solving/performance-improving nature of the activity"*.

The development of methodologies or languages in not sufficiently explored research topics implies the necessity of putting the proposed solutions into practice in order to evaluate its results. Moreover, the conclusions reached by applying them in each research iteration are high-valuable knowledge to be used in the next research iterations.

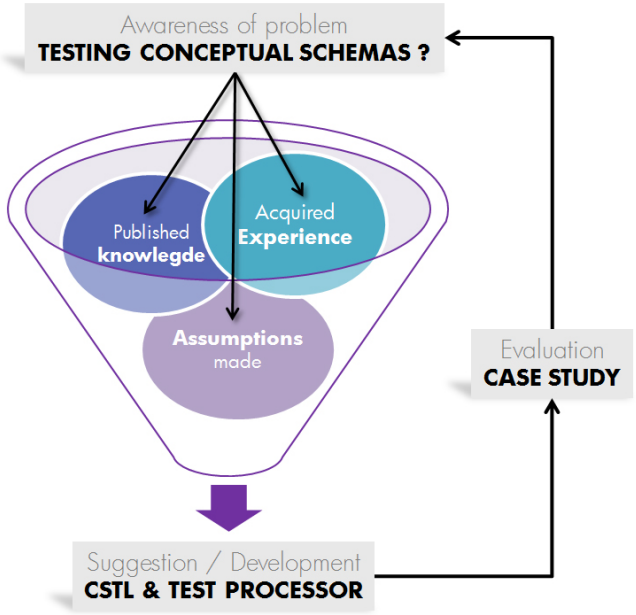


Fig. 1. The research approach based on the Design research methodology.

Figure 1 shows the main steps of the applied research approach. The starting point is the formulation of the problem to be solved: we decided to explore the idea of testing conceptual schemas of information systems in order to contribute to its quality improvement.

At the beginning, we thought about the research scope and we analyzed the viability of the research topic. Next, we defined a first version of our language and the catalog of test kinds, based on the preliminary knowledge about the problem obtained from the following sources:

- **Already published work in the research area:** We read and analyze papers, books and other publications related to the testing activity (also those relevant in other fields like programming). This analysis contributed to solidify the preliminary knowledge about the proposed research topic and was determining to analyze the viability of the research proposal.

- Once concluded that testing conceptual schemas was an interesting and open research topic, we added to the preliminary knowledge our **personal experience and background on the conceptual modeling activity** acquired during the last years in previous works.

Both the already published work and the preliminary experience, together with some initial assumptions, were the base to define a first beta-version of the CSTL language and the catalog of applicable test kinds.

In contrast with other research approaches, the design research methodology used in this work implies the development of a prototypical test processor in parallel with the definition of the proposed language and the catalog of test kinds applicable to conceptual schemas. The developed test processor allows us to put the designed language into practice in several case studies. First, we used “toy case studies” and, later on, we applied iteratively our proposal in a real-size conceptual schema of a well-known system. Some of the results of this application are shown in Section 9. It is important to note that the test processor, the catalog of test kinds and the proposed language are, at the same time, artifacts to make possible the use of design research and research results by themselves which are developed iteratively during all the research process.

Applying and exercising our language in case studies helped us accumulating knowledge about the research topic (new experience, new publications and new assumptions) which is used in order to improve the CSTL design, the catalog of test kinds and the test processor. By this way, the last iteration before the publication of this tesis report constitutes a based that we used to analyze the future work.

3. BASIC CONCEPTS AND NOTATION

3.1. The Conceptual Modeling Activity

An information system, to be useful for the people who work in a domain, must know something about this domain. Conceptual modeling is *"the activity that elicits the general knowledge that an information system needs to know about the domain and about the functions it has to perform"* [29]. An explicit conceptual schema is the specification of this knowledge and consists of a structural (sub)schema and a behavioral (sub)schema. It is elicited during the requirements engineering stage and it constitutes the basis for the system design.

Figure 2 shows a simplified fragment of the osCommerce case study focusing on shopping cart items. We use this conceptual schema fragment to illustrate the concepts explained in the following subsections.

3.2. The Conceptual Schema Under Test

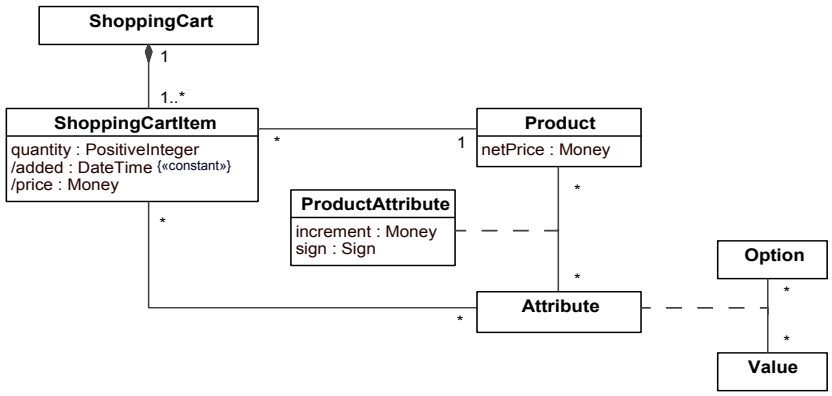
The main purpose of the Conceptual Schema Testing Language (CSTL) is providing a language to formally define test programs of conceptual schemas. The *Conceptual Schema Under Test* (CSUT) is the schema which is being tested by a given set of test programs.

In this section we are focused on reviewing the main concepts and the notation that we use to define the CSUT. We adopt the UML/OCL as the conceptual modeling language [32,33]. The adaptation of our work to other languages should be straightforward.

3.3. The Structural Schema

In the field of information systems, we make the assumption that the state of a domain can be seen as a set of objects (that we classify into concepts) and a set of relationships between them.

The specification of the set of concepts (that we call entity types) and the set of relationship types used to observe the state of a domain is called *ontology of the state* in some fields. In the field of information systems, ontologies of the state are called *conceptual schemas* of the state or simply, the *structural schema* [29].




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

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

context ShoppingCartItem::price:Money
derive:
    if self.attribute -> isEmpty() then self.product.netPrice
    else
      self.attribute.productAttribute -> select(pa|pa.product = self.product) -> collect
         (if sign = Sign::plus then increment
          else increment   -- there is an intentional typo: it should be -increment
          endif) -> sum() + self.product.netPrice
    endif

context ShoppingCartItem::added:Datetime
derive: Now()
```

Fig. 2. Fragment of the osCommerce case study focusing on shopping cart items.

The structural schema consists of a taxonomy of entity types (a set of entity types with their generalization/specialization relationships and the taxonomic constraints), a set of relationship types (either attributes or associations), the cardinality constraints of the relationship types, and a set of other static constraints formally defined in OCL. In Figure 2 there are two constraints defined as invariants in OCL: `onlyOneAttributePerOption` and `productHasTheAttributes`.

Entity and relationship types may be base or derived. The population of the base entity and relationship types is explicitly represented in the Information Base (IB). If they are derived, there is a formal derivation rule in OCL that defines its population in terms of the population of other types. Figure 2 shows an example: attribute *ShoppingCartItem*::*price* is derived; its derivation rule is given at the bottom of the Figure.

A particular class of derived relationship type that appears many times in many conceptual schemas is the constant relationship type, whose instances can be derived when the instances of one of its participants is created, and they remain fixed during its lifetime [26]. Figure 2 shows a simple example: attribute *ShoppingCartItem*::*added*. Its value is determined when an instance of *ShoppingCartItem* is created (using the derivation rule shown at the bottom of the Figure), and that value does not change later on.

Entities and relationships (that is the instances of entity types and relationship types) are considered to be concrete knowledge. The representation of the set of entities and relationships in a concrete moment of time is called the state of the *Information Base (IB)*.

3.4. The Behavioral Schema

The state of the domain is a static view of the main concepts of the system and its relationships. But the state can be queried and changed over time. The valid changes in the domain state and the actions that the system can perform are specified in the behavioral schema.

The behavioral schema consists of a set of event types. We adopt the view that events are similar to ordinary entities and, therefore, that events can be modeled as a special kind of entities, which we call event entities [28]. An event entity is an instance of an event type. There are several kinds of event types but we only deal with domain event types (which define which changes in the IB are permissible) and predefined queries (which define the information that can be requested). The adaptation of our work to languages that view events as invocations of system operations should be straightforward [19].

Modeling events as entities allows that event types, like any other entity, may have characteristics, constraints and effects. The characteristics of an event are the set of relationships in which it participates. The constraints are the conditions that events must satisfy to occur. An event constraint involves the characteristics and the state of the IB before the event occurrence. An event may occur in the state *S* of the IB if *S* satisfies all constraints and the event satisfies its event constraints. Each event type has an operation called *effect*() that gives the effect of an event occurrence. The effect is declaratively defined by the postcondition of the operation.

For domain event types, the postcondition defines the state of the IB after the event occurrence. It is assumed that the state of the IB after the event occurrence also satisfies all constraints defined over the IB. Therefore, the effect of a domain event is a state that satisfies the postcondition and all IB constraints. The method of the *effect*() operation is a procedure that produces the effect. A method is correct if the result it produces satisfies the postcondition and the IB constraints. UML does not include any particular language for writing methods. In the work reported here, we have written the methods of the *effect*() operations using a subset of the CSTL, the language we propose in this master thesis for defining tests of conceptual schemas. For queries, the postcondition of the *effect*() operation defines the answer to the query, and thus the method is not needed.

4. TESTING CONCEPTUAL SCHEMAS

4.1. Schema Testing vs. Software Testing

As we seen before, a conceptual schema specifies both the general static and dynamic knowledge required by the information system to perform its functions. In other words, it can be seen as a specification of the functional requirements of the system.

The quality of a conceptual schema comprises at least two properties: completeness and correctness. The 100% principle [15] defines completeness by stating that *"all relevant general static and dynamic aspects, i.e. all rules, laws, etc., of the universe of discourse should be described in the conceptual schema. The information system cannot be held responsible for not meeting hose described elsewhere, including in particular those in application programs"*. A conceptual schema is correct if *"the knowledge that it defines is true for the domain and relevant to the functions that the system must perform"* [29].

Since most of the work in conceptual modeling advances assume that conceptual schemas can be specified in an executable form, we are able to test them. And since the quality of the conceptual schema will contribute to the quality of the resultant information system, testing of conceptual schemas acquires real sense.

Testing conceptual schemas has some similarities with software testing. But it has also important differences. On the one hand, testing conceptual schemas does not test code, but also a technology-independent model of the system that can be executed. Moreover, the conceptual testing activity takes place in the requirements engineering stage and, therefore, we don't need to have the implementation of the system to test its functional requirements. It is interesting to remark at this point, that premature errors detection as early as possible usually reduces the costs of the entire software development process.

On the other hand, most of the work in software testing assumes that the system under test (SUT) consists of programs (objects, components) that provide only a set of operations. Testing a SUT means calling those operations with appropriate context and input parameters and checking that they return the expected outputs. For example, the recent UML Testing Profile (UTP) is based on this assumption [3,31] and the same happens in popular testing frameworks like JUnit.

If a conceptual schema were like an ordinary program, then its testing would not be very different from testing a program. However, a conceptual schema is knowledge or, more precisely, it is the general knowledge that an information system needs to know about the domain and about the functions it has to perform [29]. An executable conceptual schema can be considered a program only when there is a general-purpose information processor (virtual machine) able to behave according to the structural and behavioral rules defined in the conceptual schema [15]. Consequently, we may find some similarities between testing a program and testing a conceptual schema.

Testing conceptual schemas is as important as testing programs in projects that follows OMG's Model Driven Development (MDD) approach [30,36] when the transformation from Platform Independent Models (PIM) to Platform Specific Models (PSM) is fully automatic. This requires complete conceptual schemas, that is, conceptual schemas that include all structural and behavioral aspects.

However, we have found that it makes sense to test also incomplete conceptual schemas, as means to increase their quality [20]. Even small fragments consisting of a few entity and relationship types, integrity constraints and derivation rules can be tested to uncover their faults. This fact lays the ground for a future development of a test-driven conceptual modeling methodology, similar to the popular Test-Driven Development [4].

4.2. Functional Requirements and Tests

The main purpose of requirements elicitation activity is defining explicitly what the system should be able to perform. This is a non-trivial question. The answer implies a negotiation and prioritization process that involves all the stakeholders of the system, sometimes with several points of view and desires.

The majority of the requirements engineering methodologies advise analysts to explicitly specify how requirements will be validated. In other words, we should specify the criteria that the will be use to validate them in order to accept the resultant system. Usually, requirements validation criteria are defined using the natural language.

Tests of conceptual schemas are, in fact, concrete scenarios of functional requirement validation criteria. If defined in a formal and executable language like the one presented in this work, they can be automatically executed on the conceptual schema. By this way, we can check automatically and as many times as needed if the conceptual schema meets the requirement validation criteria in the defined cases. Martin et al. state that "*writing tests is an effective way to test requirements"* and *"writing requirements and testing are interrelated, much like the two sides of a Möbius strip"* [21].

5. RELATED WORK

As far as we know, the idea of testing conceptual schemas has not been explored in deep yet. However, we can find several related work publications in the literature that constitute a starting knowledge base of the ideas we propose.

In this section, we present some of the main references in order to give a general overview about the published work related to the topic of this masther thesis. Furthermore, in the following sections we extend it by providing references that can be useful to understand the contributions of this proposal.

5.1. CASE tools

The number of commercial CASE (Computer Aided Software Engineering) tools that help specifying conceptual schemas has increased in the last years.The most well-known commercial CASE tools (Poseidon, Magic Draw, Rational Rose, etc.) help drawing conceptual schemas written in UML but they offer rather limited verification and code-generation functionalities. Most of them perform syntactic checking on the models to prevent conceptual modelers from violating some of the well-formedness rules defined in the UML specification [33]. They also provide some functionality to generate parts of the implementation of the UML static schema. However, in general, there is a lack of support for implementing operations defined in conceptual schemas and for handling its OCL constraints and derivation rules [8].

5.2. The challenge of executing conceptual schemas

Model Driven Development [22] is a proposal adopted by the Object Management Group that proposes the development of information systems based on model transformations. In summary, MDD proposes obtaining executable software from two kinds of models: Platform Independent Models (PIM) and Platform Specific Models (PSM). The transformations between models can be manual or automatic.

Given that a conceptual schema should contain all the general knowledge about the domain which is necessary to perform its functions [29] and that we can explicitly specify it using the

standard modeling languages UML/OCL [32,33], the challenge of defining executable technology-independent models has been proposed by several researchers.

Olivé [27] proposed a research agenda to achieve the challenge of Conceptual Schema-Centric Development (CSCD). Conceptual Schema-Centric Development (CSCD) reformulates the historical goal of automating Information Systems (IS) building. CSCD emphasizes that the system's conceptual schema should be the center of the development. In this approach, the Conceptual Schema (CS) becomes the only external description to be defined. It can be executed in the production environment by using a virtual machine or by an automatic translation into software components. To achieve this goal, CSCD requires conceptual schemas to be explicit (written in a formal modeling language), complete, executable and correct. The OO-Method group directed by Dr. Pastor proposed a similar approach named eXtreme Conceptual Modeling (XCM) [17].

Testing conceptual schemas as proposed in this masther thesis requires conceptual schema executability. None of the existing commercial CASE tools allows it yet. However there are specialized tools and research prototypes that target this challenge. The USE tool [6,14,38], developed by the University of Bremen lead by Dr. Gogolla can be considered a precursor of CSTL tools. In our work, USE has been adapted to be the base for defining executable conceptual schemas to be tested. This tool is able to execute a conceptual schema written in a subset of UML and OCL. It allows creating possible system states and checking whether those states are valid instantiations of the schema. Some other tools like [11] and [9] are aimed to support some of these ideas. USE also provides the ASSL language [14] inspired on the action semantics, but it is not possible to write assertions about the state.

5.3. The testing activity

Increasing the quality of the developed information systems is the main goal of software engineering and testing is an important activity in this way [1,15]. Several researchers in conceptual modeling have proposed several verification and validation techniques in order to increase quality. In this work, we are focused on exploring the idea of testing conceptual schemas.

Most researches conclude that the main purpose of testing is exercising the developed information system by trying to make it fail [23,42]. Until now, the main well-known efforts related to the testing activity have been focused on testing code. Several tools or plugins for existing CASE tools emerged to facilitate this task. JUnit [12] is a remarkable reference tool which provides an easy-use framework to support unit testing for Java programs. Since JUnit, other unit testing tools appeared to support other programming languages [3].

Moreover, some researchers proposed methodologies to guide the testing activity like Test-Driven Development [4,18] which is based in eXtreme programming techniques.

5.4. Testing models

In the context of MDD, several researches have published some proposals to introduce approaches for testing models. Most of the efforts are focused on automatic generation of unit tests from design models. The MODEST method [24] proposed by Santos Neto et al., the Test-Driven Modeling approach [44] by Zhang and the TOTEM approach [7] by Briand an Labiche are examples of these efforts. Researchers at the Colorado State University also proposed an approach for generating tests for UML design models in order to uncover inconsistencies [10,37].

Recently, some other researchers argue that requirements specifications could also be tested. Ostroff and Ahmadi from the York University in Canada said that *"customer requirements and design specifications should be testable and testable early in the design cycle leading to early detection of requirement and specification errors"* [35]. Vrandecic and Gangemi also *"take a look at the benefits of unit testing applied to ontologies, i.e. their possibilities to facilitate regression tests"* [43]. Others like Martin et al. analyze the relationship between tests and requirements [21].

For the purpose of this work is also remarkable the publication of the UML Testing Profile (UTP) [31] by the Object Management Group (OMG). UTP is a metamodel that extends UML with test specific concepts. The terminology proposed in our work is inspired in the testing concepts defined in the UTP specification.

6. TEST KINDS APPLICABLE TO CONCEPTUAL SCHEMAS

In this work, we adopt UTP's terminology and consider that a test case is a "*specification of one case to test the system including what to test with, which input, result, and under which conditions....A test case always returns a verdict."* The verdict may be pass, fail and error [3]. In general, we consider that the verdict is Error when the conceptual schema or the test case is ill-formed (is not a valid instance of the corresponding metaschema).

When we test a conceptual schema, a test case includes one or more of the following test kinds:

- Check that a given IB state is consistent.
- Check that a given IB state is inconsistent.
- Check the contents of a given IB state.
- Check that a domain event may not occur in a given IB state.
- Check that a domain event may occur in a given IB state.
- Check that a predefined query produces the expected results

In this master thesis we propose the Conceptual Schema Testing Language (CSTL) that allows defining this kinds of tests. In the following, we first explain what we mean by IB state. Then we explain each of the above test kinds. We also include a brief description of the main CSTL constructs used to write each test kind, although the language constructs are explained in detail in the next section where the CSTL is completely specified.

6.1. Information Base State

All test kinds involve an IB state that must be specified by the conceptual modeler. This is done by indicating a set of instances of entity and relationship types [22]. We assume multiple classification and therefore we allow that an entity is instance of two or more entity types not related by generalization/specialization relationships. In UML, the instances of entity types, attributes and associations can be graphically shown [33], but in CSTL we find it more practical to use a textual notation. Figure 3 shows an example instantiation of Figure 2 in CSTL.

```
color := new Option;
shirtSize := new Option;
extraLarge := new Value;
large := new Value;
small := new Value;
largeSize := new Attribute(option := shirtSize,value := large);
smallSize := new Attribute(option := shirtSize,value := small);
extraSize := new Attribute(option := shirtSize,value := extraLarge;

fashionTShirt := new Product;
fashionTShirt.netPrice := 10;
pa1 := new ProductAttribute(product := fashionTShirt,attribute := largeSize);
pa1.increment := 5;
pa1.sign := Sign::plus;
pa2 := new ProductAttribute(product := fashionTShirt,attribute := smallSize);
pa2.increment := 2;
pa2.sign := Sign::minus;
pa3 := new ProductAttribute(product := fashionTShirt,attribute := extraSize);
pa3.increment := 7;
pa3.sign := Sign::plus;

sc := new ShoppingCart;
sci1 := new ShoppingCartItem;
sci1.shoppingCart := sc;
sci1.product := fashionTShirt;
sci1.attribute := largeSize;
sci1.quantity := 1;

sci2 := new ShoppingCartItem;
sci2.shoppingCart := sc;
sci2.product := fashionTShirt;
sci2.attribute := smallSize;
sci2.quantity := 1;
sc2.price := 8;
```

Fig. 3. An example instantiation of Figure 2.

In a test case, we define that *entityID* is a new instance of the entity types $EntityType_1,...,EntityType_n$ with the statement:

```
entityID := new EntityType1,...,EntityTypen;
```

To define that the value of attribute *att* of entity *entityID* is *val* (where *val* is a valid OCL expression) we write:

```
entityID.att := val;
```

Similarly, to define that the entity *entityID* is related with role *r* in a binary link (an instance of association) to one or more entities given by the OCL expression *participants* we write:

```
entityID.role := participants;
```

Instances of an n-ary UML association *Assoc* with roles $r_1,...,r_n$ are created with the statement:

```
new Assoc(r1:= entityID1,..., rn:= entityIDn);
```

If *Assoc* is an association class, then the above statement returns the identifier of the instance of that class.

An important distinction must be made between the base and derived parts of an IB state. We call *base state* the subset of a state comprising the instances of base types explicitly specified by

the conceptual modeler. Derived constant relationship types must be considered as base types in this respect because they are derived only at creation time.

The *derived state* is the subset of the state comprising the instances of derived types, as specified by their corresponding derivation rules. The derived state can be computed by the system when it is needed. However, we have found that, for testing purposes, a conceptual modeler may wish to explicitly define one or more instances of derived types. We call *materialized state* the subset of the state comprising the instances of derived types explicitly given by the conceptual modeler. When the materialized state is consistent with the derivation rules, then it is a subset of the derived state. However, this does not happen when the materialized state is inconsistent. Note that in Figure 2 attribute *ShoppingCartItem*::*price* is derived and that in the IB state of Figure 3 it has been instantiated for *sci*2, but not for *sci*1.

6.2. Check that a given IB state is consistent

Using CSTL, in order to check that the current IB state is consistent, the conceptual modeler just writes the statement:

```
check consistency;
```

The result is Pass if the IB state is consistent; and Fail otherwise.

Checking consistency includes two steps: checking the static constraints and checking the materialized state. Checking that an IB state satisfies the static constraints is well known. It can be done by considering each static constraint in turn and checking that it is satisfied by the base state under test. Note that the materialized state is ignored in this step. If checking a constraint requires the population of the derived state, then it is computed. Obviously, the verdict of the test case is Pass if the base state satisfies all static constraints, and Fail otherwise.

As far as we know, the problem of checking the materialized state has not been considered in the literature. An arguably necessary condition is that the materialized state is a subset of the derived state. But this condition may not be sufficient. We tend to believe that if there are one or more instances of a derived type in the materialized state, then we must assume that all instances of that type are in that state. This is an *all or nothing* assumption: either the conceptual modeler specifies the full population of a derived type or none at all.

We check the materialized state in the following way. Let *D* be a derived type that has one or more instances in the materialized state. Then:

1. Transform type *D* into a base one. The corresponding materialized state becomes base.
2. Transform the derivation rule of *D* into a constraint.
3. Check the constraint.

4. Undo the changes (1) and (2).

If we check the consistency of the IB state shown in Figure 3 the result is Fail. All static constraints, including the two invariants shown in Figure 2, are satisfied, but the materialized state is inconsistent for two reasons: (1) the derived price for *sc*2 is 12 instead of the expected 8, and the value of *sc1.price* is missing. The test can Pass if the derivation rule of *ShoppingCartItem*::*price* is corrected as indicated in Figure 2, and the following statement is added to Figure 3:

```
sc1.price := 15
```

Generally, a conceptual modeler writes this kind of test to check that (1) the structural schema can be instantiated to represent a particular domain state; (2) the whole set of constraints and derivation rules defined in the schema behave as expected; and (3) the set of constraints defined in the schema is strongly satisfiable (because there is at least one non-empty IB state that satisfies them).

6.3. Check that a given IB state is inconsistent

This kind of test is the inverse of the previous one. To check that the current IB state is inconsistent, the conceptual modeler just writes the statement:

```
check inconsistency;
```

The verdict is Pass if the IB state is inconsistent; and Fail otherwise.

For example, the conceptual modeler may wish to test that the invariant *onlyOneAttributePerOption* does not allow an item with a product having two attributes of the same option. He or she may change the assignment of *sc1.attribute* in Figure 3 to:

```
sci1.attribute := largeSize,smallSize;
```

Now the test of *check inconsistency* will Pass because *sci*1 has a product (*fashionTShirt*) that is both *large* and *small*.

A conceptual modeler writes this kind of test to check that (1) the OCL constraints behave as expected; or (2) the whole set of constraints and derivation rules defined in the schema behave as expected.

6.4. Check the contents of a given IB state

This kind of test checks whether the current state of the IB satisfies a boolean condition defined in OCL. The conceptual modeler just writes the statement:

```
assert true booleanExpression;
```

where *booleanExpression* is an OCL expression over the types of the IB and the variables of the test case. The verdict is Pass if *booleanExpression* is true, and Fail otherwise. If the current state is inconsistent, then the verdict is Error. Other CSTL statements for this kind of test are: *assert false, assert equals* and *assert not equals*.

For example, the conceptual modeler may wish to test that the prices of the two shopping cart items are different, even if they buy the same product (but have different attributes). He or she may write the statement:

```
assert false sci1.price = sci2.price;
```

In the IB state of the Figure 3 the verdict of this test is Pass.

A conceptual modeler writes this kind of test to check that (1) the structural schema can be instantiated to represent a particular domain state; (2) one or more derivation rules derive the expected results; (3) a navigational expression yields the expected results; or (4) the effect of one or more domain events -see below- implies an expected result on the IB.

The usefulness of checking that a structural schema can be instantiated to represent a particular domain state was observed already 20 years ago in the NIAM method, which included it as a *population check*: "being able to easily populate a conceptual schema diagram is useful not only for detecting schema diagrams that are non-sensical, but also for discussing constraints" [25:50]. Using CSTL and a test processor, the conceptual modeler can easily write tests that populate conceptual schemas, and automatically execute such tests as often as needed.

6.5. Check that an event or query may not occur

Domain event types and predefined queries may have constraints. The meaning is that the instances of those types or queries may only occur in the domain if the constraints are satisfied. This kind of test checks that a particular domain event or query may not occur because its constraints are not satisfied in the current state of the IB. This kind of test is similar to the *violation test case* of E-Tester [34]. In CSTL, to test that a domain event or query with characteristics $c_1,...,c_n$ may not occur the conceptual modeler writes the statement:

```
new [DomainEventType|Query] (c1:= expression1,..., cn:= expressionn) may not
occur;
```

where *expression*$_i$ is an OCL expression over the types of the IB and the variables of the test case. The verdict is Pass if the constraints of the domain event or query are not satisfied. If the event or query constraints are satisfied, the verdict is Fail, and the event or query produces no effect. In all cases, if the current state is inconsistent, then the verdict is Error. Note that if the domain event type or query does not have constraints then the verdict will be Fail because nothing prevents them to occur (provided that the current state is consistent).For example, consider the domain event type *DeleteProductAttribute*, with characteristic an instance of *ProductAttribute*, which corresponds to the fact that in the domain a given product ceases to be available in a given attribute (option/value). Assuming that we have not defined (yet) constraints for this event, if we want to check that *pa*1 (see Figure 3) cannot be deleted, we write:

```
new DeleteProductAttribute(productAttribute:=pa1) may not occur;
```

The verdict is Fail. We must define an event constraint that prevents the occurrence of the event when the product attribute to be deleted is used in some shopping cart item.

A conceptual modeler writes this kind of test to check that (1) the OCL event constraints behave as expected; and (2) the whole set of constraints defined in the event or query does not allow its occurrence as expected.

6.6. Check that a domain event occurs

This kind of test checks that the effect of a domain event occurrence is as expected. The conceptual modeler writes the statement:

```
domainEventID := new DomainEventType(c1:= expression1,..., cn:= expressionn)
occurs;
```

where, as before, *expression*$_i$ is an OCL expression over the types of the IB and the variables of the test case. This checking includes the following steps:

1. Check that the current IB state satisfies the static constraints. The verdict is Error if any of the constraints is not satisfied.
2. Check that the constraints of the event are satisfied. The verdict is Fail if any of the event constraints is not satisfied.
3. Execute the method of the corresponding *effect*() operation.
4. Evaluate the derivation rules of the derived constant attributes and associations for the entities created in the previous step, and store their value in the IB. The check that the result of these derivation rules is as expected can be made later on, using the test kinds described in 6.4 and 6.7.

5. Check that the event postconditions are satisfied. The verdict is Fail if any of the postconditions is not satisfied.
6. Check that the new IB state satisfies the static constraints. The verdict is Fail if any of the constraints is not satisfied; otherwise the verdict is Pass.

For example, assume that the postcondition of the domain event type *DeleteProductAttribute* has been defined as:

```
context DeleteProductAttribute::effect()
  post: not productAttribute@pre.oclIsKindOf(OclAny)
```

Then, the test:

```
dpa:=new DeleteProductAttribute(productAttribute:=pa3) occurs;
```

gives the verdict Pass if the method of the operation correctly deletes *pa*3.

Note that the standard OCL allows references to the state of the IB prior to the event occurrence (@pre) in the postcondition. Therefore, in contrast with the proposal [22], in CSTL we do not need to refer to that state.

A conceptual modeler writes this kind of test to check that (1) the OCL event constraints behave as expected; (2) the whole set of constraints defined in the event behave as expected; and (3) the method and the derivation rules of the derived constant attributes and associations produce the expected results (satisfaction of postconditions and static constraints).

6.7. Check that a predefined query produces the expected results

This kind of test checks that a predefined query may occur. The conceptual modeler writes the statement:

```
queryID := new Query (c1:=expression1,..., cn:=expressionn)    occurs;
```

where, as before, $expression_i$ is an OCL expression over the types of the IB and the variables of the test case. If the current state is inconsistent, then the verdict is Error. The verdict is Fail if any of the constraints of the query is not satisfied. If the query constraints are satisfied, the verdict is Pass. Note that in this case the postconditions of the query are not checked, because there is no method.

A conceptual modeler writes this kind of test to check that: (1) the query constraints behave as expected; (2) the effect of one or more previously occurred domain events has produced the expected results on the IB; and (3) the postcondition of the query gives the expected results.

7. CSTL SPECIFICATION

In this section we present a detailed specification of the Conceptual Schema Testing Language (CSTL) and the principles of its design.

7.1. Five Design Principles of CSTL

The essential purpose of CSTL is providing a textual, procedural, formal and executable notation for writing automated tests of conceptual schemas written in UML/OCL [32,33].

CSTL is not merely a notation to make possible to write tests of conceptual schemas, but also it facilitates, by itself, the testing activity.

CSTL syntax has been designed by finding a balance between expressiveness, simplicity and understandability of the specified tests. In order to achieve this purpose, CSTL design is based in the following principles:

- **CSTL allows defining the tests kinds applicable to conceptual schemas** defined in the previous section.

- **CSTL facilitates the task of writing tests**. Given that writing tests consume time, CSTL pretends to make possible the definition of tests guided by the idea to express as much information as possible by writing as less as possible. In other words, we find a balance between simplicity and expressiveness. This objective is more feasible in a specialized language like CSTL than in a general purpose language.

- **CSTL is focused on enhancing tests understandability:** Tests of executable conceptual schemas specified in CSTL can be seen as executable specifications of concrete requirements scenarios. CSTL tests, once defined, have the particularity that they can be executed automatically as many times as needed. Consequently, they are an interesting approach for validating requirements. In this context, CSTL syntax has been designed to be easy understandable and as closed as possible to the way of describing tests in the natural language. The definition of associated pattern sentences to each language statement was a key technique to guide the CSTL design.

- **CSTL follows the style of the modern xUnit testing frameworks:** CSTL syntax is inspired on existing languages that are related to CSTL objectives or used for testing in

other context and fields, but not adaptable or suitable at all to test conceptual schemas. CSTL is based on the style of xUnit [12] testing languages in the field of programming. It also incorporates the OCL syntax to navigate through the conceptual schema under test.

- **CSTL tests can be executed by an interpreter:** The proposed language has been designed as an automatically executable language. We developed an interpreter that makes possible to execute tests written in CSTL.

CSTL has been inspired in, and is an evolution of, USE and ASSL [14,38]. CSTL is able to deal with richer conceptual schemas because: (1) it allows derived entity and relationship types; (2) in particular, it allows derived constant relationship types [26]; (3) events and predefined queries are conceptualized as entities and not as operation invocations [28]; and (4) it deals with conceptual schemas that allow multiple classification of entities.

7.2. Test Program Structure

Figure 4 shows the fragment of the metamodel of *test programs*. A test program is the top-level grouping structure of CSTL tests. It consists of:

- A set of **test cases**: A test case is a "specification of one case to test the system including what to test with, which input, result, and under which conditions" [31]. A test case includes one or more of the test kinds applicable to conceptual schemas that we enumerated in the previous section.

- A **fixture**: The fixture is a set of statements that create a state of the IB and define the values of the common program variables. It is assumed that the execution of each test case starts with an IB state and the contents of the variables as defined by the fixture. By this way, we can create a common initial state configuration that is shared by all the test contexts included in the test program. The fixture of a test program can be empty.

- A set of **fixture components**: A fixture component is a named set of statements that create a fragment of the state of the IB and define the values of a set of variables. In contrast with the program fixture, fixture components must be load explicitly in test cases or in the program fixture when needed.

Fixture components allow us to create IB state configurations than can be selectively applied. The set of fixture components may be empty.

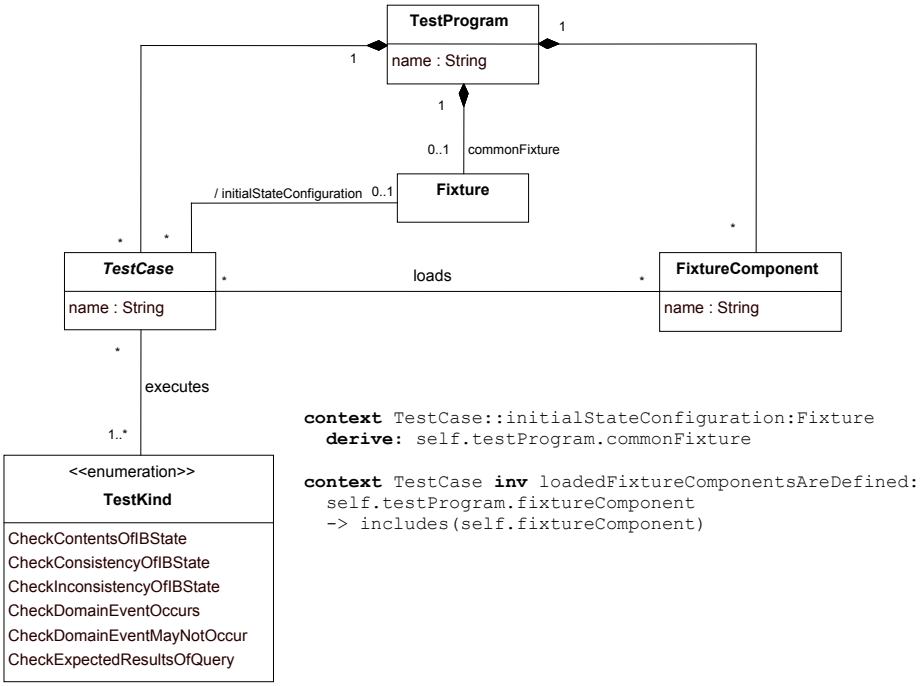


Fig. 4. CSTL metamodel fragment of test programs.

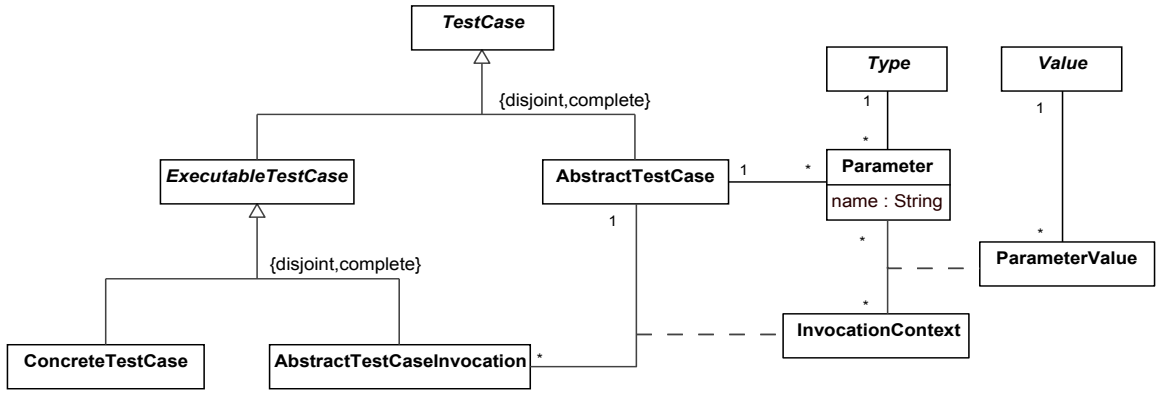
7.3. Kinds of Test Cases

CSTL allows specifying three kinds of tests:

- **Concrete test case:** A concrete test case is an executable set of statements that builds a state of the IB, define and assign values to variables and executes one or more test kinds.

- **Abstract test case:** An abstract test case is a parameterized test case that can be invoked several times in a test program. An abstract test case cannot be executed.

- **Abstract test case invocation:** Abstract test cases can be invoked by giving a concrete context (defined by the desired values assigned to parameters). Abstract test case invocations admit fixture components as parameters. By this way, this kind of test case allows testing the same tests in different IB configurations.




```
context InvocationContext inv valuesForEachParameter:
   slef.parameter->size() = self.abstractTestCase.parameter->size()
```

Fig. 5. CSTL metamodel fragment of test cases.

Each test case is independent. The independence of test cases is one of the main characteristics of the xUnit code testing frameworks. It means that after the execution of each test case, the IB is reset and it comes back to the state defined by the fixture of the test program that contains the test case.

7.4. CSTL Types and Value Expressions

CSTL allows the value types defined in the OCL 2.0 metamodel [32]. Morevoer, the language introduces a specific type called *FixtureComponentType*. This specific type permits declaring fixture components and using them as parameters for abstract test cases.

CSTL permits the use, as values, of the different kinds of *ValueSpecifications* defined in the UML 2.0 metamodel [33]. A fixture is also a valid value in the context of CSTL.

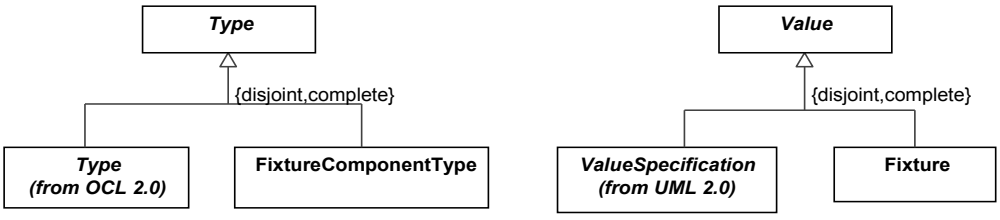
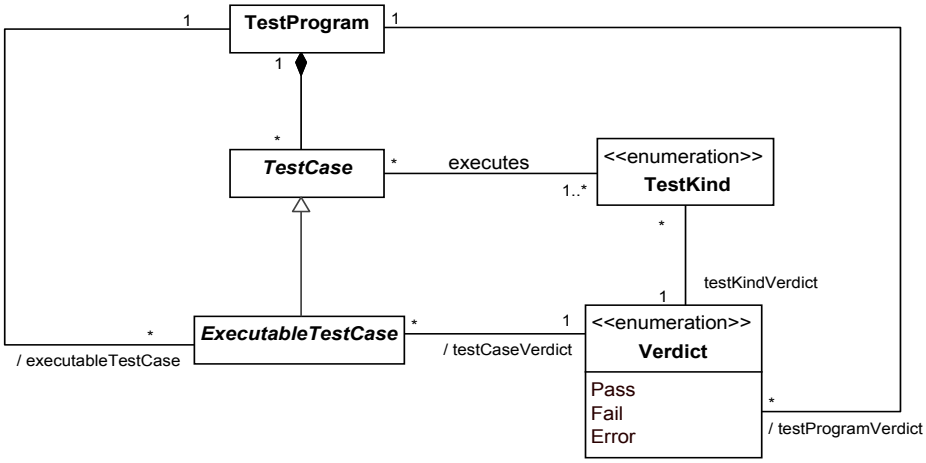


Fig. 6. CSTL metamodel fragment of CSTL values and types.

7.5. Test Verdicts

The execution of a test case gives a **Verdict** as a result. Verdict values can be *Pass*, *Fail* or *Error*. The verdict of a test case is obtained from the verdicts of the test kinds it executes. Test programs also have a verdict as a composite result of the test cases it groups.

Figure 7 shows the fragment of the CSTL metamodel corresponding to verdicts. Note that the derivation rules specify how test cases verdicts and test program verdicts are obtained from the verdicts of the execution of each test kind. Section 6 explains in detail how test kind verdicts are obtained.




```
context TestProgram::executableTestCase:ExecutableTestCase
derive:
   self.testCase->select(tc | tc.oclIsTypeOf(ExecutableTestCase))


context ExecutableTestCase::testCaseVerdict:Verdict
derive:
   if   self.testKind.testKindVerdict -> includes (Verdict::Error)
   then Verdict::Error
   else
      if    self.testKind.testKindVerdict -> includes (Verdict::Fail)
      then  Verdict::Fail
      else  Verdict::Pass
      endif
   endif


context TestProgram::testProgramVerdict:Verdict
derive:
   if   self.executableTestCase.testCaseVerdict ->includes (Verdict::Error)
   then Verdict::Error
   else
      if  self.executableTestCase.testCaseVerdict -> includes (Verdict::Fail)
      then  Verdict::Fail
      else  Verdict::Pass
      endif
   endif
```

Fig. 7. CSTL metamodel fragment of test verdicts.

7.6. Language syntax

In the previous sections we explained the semantics of the main elements of CSTL. In this section we present the CSTL syntax used for defining test programs and test cases by means of the top-level fragment of the CSTL grammar. The syntax and the semantics of the leaf statements of this grammar fragment are detailed in the following section.

```
testProgram
    : testprogram <programName> { fixture? fixtureComponent* testCase* }

fixture
    : (stateStatement ;)+

fixtureComponent
    : fixturecomponent <fixtureComponentName> { (stateStatement;)+ }

testCase
    : concreteTest
    | abstractTest
    | abstractTestInvocation

concreteTest
    : test <testName> { (testStatement ; | controlFlowStatement)* }

abstractTest
    : abstract test <testName> (parameter*) { (testStatement ; | controlFlowStatement)* }

parameter
    : type <parameterName>

abstractTestInvocation
    : test <abstractTestName> (parameterAssignment*) ;

parameterAssignment
    : <parameterName> := value

testStatement
    : stateStatement
    | variableStatement
    | assertion

stateStatement
    : entityCreation
    | entityDeletion
    | binaryPropertySetting
    | nAryRelationshipCreation
    | fixtureComponentLoading
```

```
variableStatement
    : variableDeclaration
    | variableAssignment

assertion
    : assertTrue
    | assertFalse
    | assertEquals
    | assertNotEquals
    | checkConsistency
    | checkInconsistency

controlFlowStatement
    : conditional
    | forLoop
    | forEachLoop
    | whileLoop
```

Figure 8 shows a generic example that conforms to CSTL syntax.

```
testprogram TestProgramName{

        //FIXTURE
        //State statements located here compose the fixture

        //FIXTURE COMPONENTS
        fixturecomponent FixtureComponentName1{
          //State statements
        }

        fixturecomponent FixtureComponentName2{
          //State statements
        }

        …

       //TEST CASES

       test TestName{
         //Test instructions
       }

       abstract test abstractTestName
       (ParamType1 ParamName1, ParamType2 ParamName2,…){
         //Test instructions
       }

       test abstractTestName
       (paramName1 := paramValue1, paramName2 := paramValue2,…);

       …
```

Fig. 8. Generic CSTL program.

7.7. CSTL statements

7.7.1. State statements

We can load a state of the Information Base by executing a set of state statements. In this section we present the syntax for:

- Creating and deleting entities.
- Setting binary properties of an entity (attributes or binary relationships).
- Creating new n-ary relationships between entities.
- Loading a fixture component.

State statements can be used in fixtures, fixture components and test cases.

Entity creation

Syntax

```
[entityID :=] new EntityType1,...,EntityTypen
                [(propertyID1:=valueExpression1,..., propertyIDn:=valueExpressionn)];
```

Pattern Sentence

"An entity `entityID` is a new instance of the entity types `EntityType1,...,EntityTypen`. The value of `valueExpression1` is assigned to the property `propertyID1`,... and the value of `valueExpressionn` is assigned to property `propertyIDn`".

In some cases we don't need to use an entity in other expressions after its creation in the Information Base (IB). If the name of the new entity is not specified, it is created with an Internal Object Identifier (OID). Therefore, the entity is created, but later it cannot be referenced in other expressions.

The order in which properties are specified is irrelevant. This is an interesting characteristic of CSTL. If we add, remove or reorder properties in the *Conceptual Schema Under Test* (CSUT) we don't need to change already done tests. Moreover, properties can be attributes or binary association ends. If we change the way of representing a property, we also don't need to change the already written tests.

Note that we allow multiple classification. That is, an entity can be instance of several entity types at the same time.

Event occurrence

We adopt the approach that events are modeled in the CSUT as stereotyped entities [28]. Therefore, the basic syntax used for checking that an event (domain event, action request event or query) occurs or not is very similar to the syntax used for *entity creation*. However, the semantics are more extensive and two options are included to check if an event can occur or not.

Event entities are created as any other entity, but they specify an operation effect(), with an associated method, that is the procedure that specifies the effect of the event in an executable form. If the created entity type is an event, the following semantics are applied:

Syntax

```
[eventID :=] new EventEntityTypeID
(c1:=valueExpression1,..., cn:=valueExpressionn)
[occurs | may not occur];
```

Pattern Sentence

"The `eventID` *is a new event* `EventEntityTypeID` *(with the characteristics* c_1 *with the value* `valueExpression1`*,... and the characteristic* c_n *with the value* $valueExpression_n$*) that* `occurs|may not occur`*"*

If we don't specify any of the occurrence options (*occurs* or *may not occur)* the event effect is not applied, although the event entity is created in the IB. We can execute the effect later by writing:

Syntax

```
eventID [occurs | may not occur];
```

Pattern Sentence

"The effect of the `eventID` *occurs".*

Using this syntax variant we can create the event entity and specify characteristics of it in several separate instructions. We can specify the event characteristics by using the state instructions (as it is done for any other entity). Once we have specified all the required characteristics, we can explicitly indicate the effect execution.

An event may not occur if the constraints are not satisfied in the IB state. If all the constraints are satisfied, the event can occur and the verdict of the associated test is Pass.

Note that the order in which we specify the event characteristics is irrelevant.

Entity deletion

Syntax

```
delete entityID;
```

Pattern Sentence

"Delete the entity `entityID`*"*

Binary property setting

Syntax

```
entityID.role := participant1,...,participantn;
```

Pattern Sentence

"The entity `entityID` *is related with role* `role` *to the entities* `participant1,...,participantn`*"*

Note that this statement can be used for assigning UML attributes or association ends. CSTL considers that an entity has binary properties regardless how they are expressed in UML (as an association or as an attribute). This is a remarkable characteristic of CSTL if used in a test-driven conceptual modeling environment in which tests are written incrementally [40]. This abstraction avoids changing the already done tests if we decide to change the way of representing a binary property.

N-ary relationship creation

Syntax

```
new AssociationID := (roleID1 := entityID1,..., roleIDn := entityIDn) ;
```

Pattern Sentence

"The association `AssociationID` relates the entitiy `entityID1` with the role `roleID1` ,..., and the `entityIDn` with the role `roleIDn`"

This statement requires two or more entities to be related (n>2). For n=2, the *Binary property setting* can be applied with the same result in the IB state.

The order in which we assign entities to roles is irrelevant and it does not depend on the order in which they are specified in the CSUT.

Fixture component loading

Syntax

```
load fixtureComponentID ;
```

Pattern Sentence

"Load the IB state changes as specified by the fixture component `fixtureComponentID`"

The loading process executes the state instructions specified by the fixture component. Therefore, the IB state is modified as indicated by the state instructions that constitute the loaded fixture component.

7.7.2. Variable statements

CSTL allows storing values in variables to be used in subsequent statements. In this section we present the syntax for declaring variables and for assigning values to these variables.

Variables are only visible in its scope which is determined by the location in which they are declared. The scope of a variable is the structure (test program, fixture component, test case, or control flow statement) where it has been declared and its nested substructures.

Variable declaration

Syntax

```
varType varID ;
```

Pattern Sentence

"The variable `varID` of type `varType` is declared"

Note that variable declaration it can be used for declaring a variable in the desired context (in order to make it visible in the corresponding structures) with an undefined value.

Variable assignment

Syntax

```
varID := valueExpr;
```

Pattern Sentence

"The resulting value of the expression `valueExpr` is assigned to the variable `varID`".

If the variable `varID` is not declared, the statement becomes a *VariableAssignmentAndDeclaration.*

A value expression is an OCL expression evaluated on the current state of the IB.

Variable assignment and declaration

Syntax

```
[varType] varID := valueExpr;
```

Pattern Sentence

"The resulting value of the expression `valueExpr` is assigned to the new variable `varID` [of type `varType`]".

A value expression is an OCL expression evaluated on the current state of the IB.

Note that this is a composite statement. It allows declaring a new variable and assigning a value to it with only one statement.

If the varType is not specified, it is assumed that the type of the new variable corresponds to the predefined type of the assigned value expression.

7.7.3. Assert statements

Assert statements allow formalizing expected assertions about the current state of the Information Base. These assertions contribute to make the tests automatically executable. Once defined the assertions of a test, they can be checked automatically as many times as needed.

Assert true

Syntax

```
assert true booleanExpression;
```

Pattern Sentence

"Assert that the expression `booleanExpression` *is true in the current state of the IB".*

A Boolean expression is an OCL expression the result of which, after its evaluation on the current state of the IB, is a Boolean value.

Assert false

Syntax

```
assert false booleanExpression;
```

Pattern Sentence

"Assert that the expression `booleanExpression` *is false in the current state of the IB".*

Assert equals

Syntax

```
assert equals valueExpression1 valueExpression2 ;
```

Pattern Sentence

"Assert that the expression `valueExpression1` *is equal to* `valueExpression2`*".*

A value expression is an OCL expression evaluated on the current state of the IB.

Assert not equals

Syntax

```
assert not equals valueExpression1 valueExpression2 ;
```

Pattern Sentence

"Assert that the expression `valueExpression2` *is not equal to* `valueExpression2`*".*

A value expression is an OCL expression evaluated on the current state of the IB.

Check consistency

Syntax

```
check consistency;
```

Pattern Sentence

"The current state of the IB is consistent".

This statement checks that the IB satisfies all the static constraints defined in the conceptual schema under test. If before this statement, there are instances of a derived type, the materialized state corresponding to the instantiated derived types is also checked.

Check inconsistency

Syntax

```
check inconsistency;
```

Pattern Sentence

"The current state of the IB is inconsistent".

This statement checks that the IB:

- does not satisfy at least one of the static constraints defined in the conceptual schema under test, or
- the materialized state corresponding to the previously instantiated derived types is not consistent.

7.7.4. Control flow statements

Control flow statements allow altering the sequential order in which a set of statements are executed. CSTL provides conditional statements to execute alternative sets of statements depending on the evaluation of a specified condition over the IB state. CSTL also provides loop structures to automatically repeat the execution of a set of statements while a specified condition is satisfied.

Conditional statement

Syntax

```
if booleanExpression1 then statements1
[else if booleanExpression2 then statements2]
 …
[else if booleanExpressionn-1 then statementsn-1]
[else statementsn]
endif
```

Pattern Sentence

"If the expression `booleanExpressioni` evaluates true the set of statements `statementsi` is executed. Otherwise, the set of statements `statementsn` is executed".

For statement

Syntax

```
for [varType] varID := valueExpr1 to valueExpr2 step valueExpr3
do statements
endfor
```

Pattern Sentence

"Given a variable `varID` initialized with the value of $valueExpr_1$ the set of statements `statements` are repeated until `varID` is equal to the value of $valueExpr_2$. In each iteration the value obtained by evaluating the expression $valueExpr_3$ is assigned to `varID`".

If the variable `varID` has not been declared yet in the scope, it is declared automatically with the specified type. The *for statement* is the scope of the variables declared inside it.

If the variable type is not explicitly specified and the variable has not been declared yet in the scope, the variable is declared automatically with the predefined type of the assigned expression ($valueExpr_1$).

If the variable type is explicitly specified and the variable is already declared, `varType` must be of the type of the variable `varID`.

Note that value expressions should be compatible with the variable type. A type T_1 is compatible with type T_2 if values of types T_1 can be assigned to variables of type T_2.

For each statement

Syntax

```
for each [varType] varID in collectionExpr
do statements
endfor
```

Pattern Sentence

"For each element of the resultant collection of `collectionExpr` do the set of statements `statements` which can use the current element of the collection, which is stored in the variable `varID`".

`collectionExpr` is an OCL expression the result of its evaluation is a collection.

The type of the variable `varID` must be compatible with the type of the `collectionExpr`.

If the variable has not been declared yet, it is automatically declared. The already declared variable is reused to store the current element of the collection in each iteration.

If the type of the variable is not specified and the variable needs to be created, it is assumed that the type of the new value is the predefined type resulting of the evaluation of the expression `collectionExpr`.

While

Syntax

```
while booleanExpr
do statements
endfor
```

Pattern Sentence

"While `booleanExpr` evaluates true repeat the set of statements `statements`".

`booleanExpr` is an OCL expression the result of its evaluation is a Boolean value.

8. THE CSTL ENVIRONMENT

Figure 9 shows the relationship between the definition and execution of a conceptual schema and the definition and processing of its tests. We have implemented the information processor reusing USE [14] as much as possible.

We have developed a test interpreter that reads a CSTL program and executes its statements. The test interpreter coordinates the execution of the tests (setting up fixtures, computing verdicts, and so on), invokes the services of the information processor to create, remove and change entities, attributes and associations of the IB, and also to evaluate OCL expressions over the IB. Moreover, it shows the results of the test execution. The test manager stores the CSTL programs and requests their execution to the test interpreter. The test manager also keeps track of the test results, and maintains test statistics.

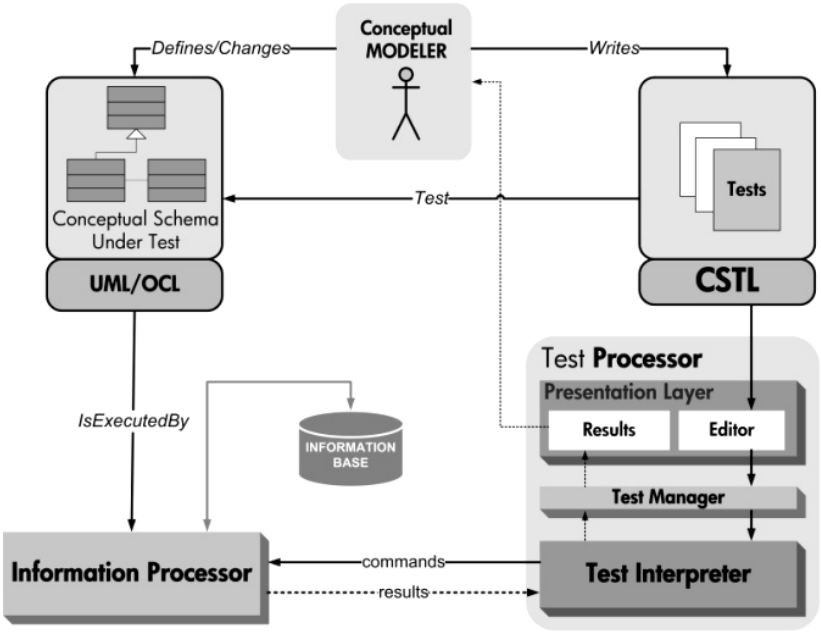


Fig. 9. Test processing and conceptual schema execution

Figure 11 shows the result of the execution of the CSTL program of Figure 10 that tests the confirmation of an order taking the information of a shopping cart (Figure 2). There are two test cases that have failed, and therefore the global verdict is Fail. Note that the test processor

indicates the number of the lines where the tests have failed, and an explanation of the failure in natural language. More examples can be found in section 9.

```
testprogram OrderConfirmation{

       //FIXTURE
       //Products and attributes initialization
       shirtSize := new Option;
       extraLarge := new Value;
       small := new Value;
       smallSize := new Attribute(option := shirtSize, value := small);
       extraLargeSize := new Attribute(option := shirtSize, value := extraLarge);

       //Products initialization
       fashionTShirt := new Product;
       fashionTShirt.netPrice := 10;
       smallFashionTShirt := new ProductAttribute
                             (product := fashionTShirt, attribute := smallSize);
       smallFashionTShirt.increment := 2;
       smallFashionTShirt.sign := Sign::minus;
       extraLargeFashionTShirt := new ProductAttribute
                             (product := fashionTShirt, attribute := extraLargeSize);
       extraLargeFashionTShirt.increment := 1;
       extraLargeFashionTShirt.sign := Sign::plus;

       //Customer shopping cart initialization
       c := new Customer;
       sc := new CustomerShoppingCart;
       sc.customer := c;

       fixturecomponent addRegularSizedTShirts{
          item1 := new ShoppingCartItem;
          item1.product := fashionTShirt;
          item1.quantity := 3;
          item1.shoppingCart := sc;
       }

       fixturecomponent addSpecialSizedTShirts{
          item1 := new ShoppingCartItem;
          item1.product := fashionTShirt;
          item1.shoppingCart := sc;
          item1.quantity := 2;
          item1.attribute := smallSize;
          item2 := new ShoppingCartItem;
          item2.product := fashionTShirt;
          item2.shoppingCart := sc;
          item2.quantity := 1;
          item2.attribute := extraLargeSize;
       }

       test emptyShoppingCart{
          check consistency;
       }

       abstract test confirmedOrderTotal (Fixture itemsAddition, Money expectedTotal){
           load itemsAddition;
           oc := new OrderConfirmation(shoppingCart := sc) occurs;
           assert equals oc.orderCreated.total expectedTotal;
       }

       test confirmedOrderTotal
            (itemsAddition := addRegularSizedTShirts,expectedTotal := 30);

       test confirmedOrderTotal
            (itemsAddition := addRegularSizedTShirts,expectedTotal := 30);

}
```

Fig. 10. CSTL program for testing order confirmation

The CSTL interpreter assumes that the CSUT is specified in the USE format. The USE syntax is explained in detail in [6]. Note that USE syntax adopts some particular notation for some UML constructs and OCL expressions. For example: data types must be specified as UML classes, enumeration values are referenced with the symbol '#' and *allInstances* expressions does not admit parethesis like in the standard OCL.

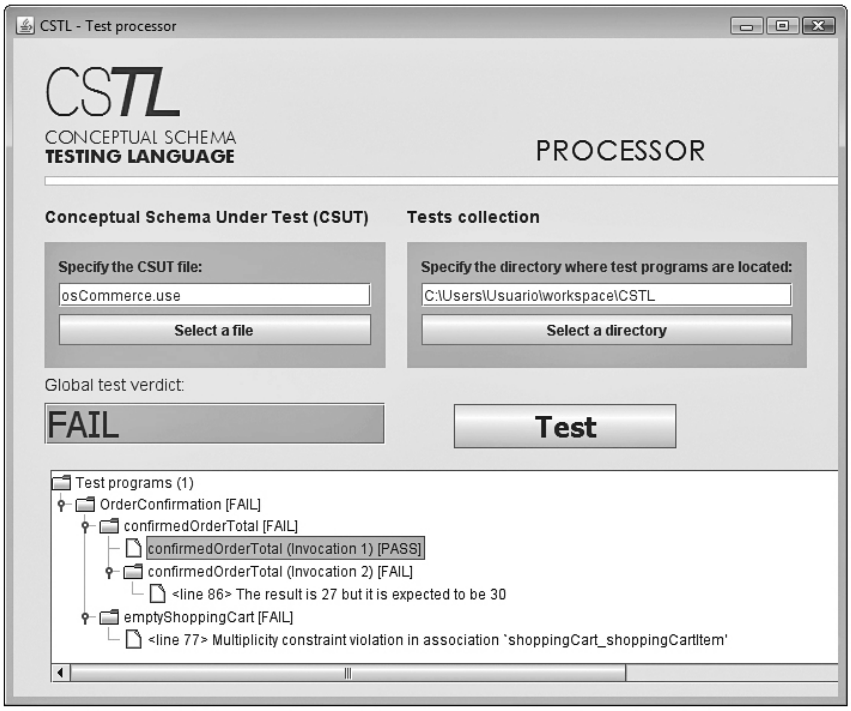


Fig. 11. Test processor screenshot

USE does not allow derived types and the definition of event constraints, that are always creation-time constraints because they must be evaluated when the event occurs. In order to allow the specification of these relevant characteristics, we enriched the USE syntax as follows:

- **Derived Types**. An attribute *Attr* is assumed to be derived if it is preceeded by the character '_'. Therefore, it is assumed that *_Attr* is a derived attribute named *Attr*. The derivation rule must be specified as an operation without parameters named *Attr()*. Consider the following class definition as an example:

```
class Category
attributes
    imagePath:String
    _subcategories:Integer -This is a derived attribute
operations
    subcategories():Integer=self.child->size()
```

- **Initial Integrity Constraints.** Creation-time constraints are also allowed by using the enriched syntax of USE used in the CSTL interpreter. This particular type of constraints can be explicitly defined by adding the string "_iniIC_" before the constraint name as indicated in the following example:

```
context OrderConfirmation inv _iniIC_ShippingMethodIsEnabled:
   self.shippingMethod.status= #enabled
```

9. APPLICATION TO THE OSCOMMERCE CONCEPTUAL SCHEMA

E-commerce allows people exchanging goods and services with no barriers of time or distance.

***osCommerce* [8]** is an e-commerce solution available as free software under the GNU (General Public License). *osCommerce* project was started in March 2000 in Germany and since then, it has become the base of thousands of online stores around the world. *osCommerce* can be customized to operate in different countries (with different languages, taxes, currencies,…) and to be used in several kinds of online stores.

In this section we provide some example test programs taking a set of representative concepts and domain events of the osCommerce conceptual schema [41] as the Conceptual Schema Under Test (CSUT).

The osCommerce conceptual schema models the real *osCommerce* system that includes a considerable number of domain concepts, relationships and events. Therefore, the schema is organized in subschemas in order to improve its comprehension. The CS of the osCommerce system models the structural knowledge of the system in UML/OCL and gives the specification of the more relevant use cases in an informal textual description. Uses cases are linked to the events wich are formally defined in UML/OCL.

We start by giving a general overview of the main concepts of the osCommerce domain. After that, example CSTL programs are presented as follows: for each substructural schema, we show the most relevant use cases that require the static knowledge represented in the substructural schema. Then, we show the most relevant associated events. And after that, we reproduce some example test programs related to them.

Given that CSTL test cases can be used to test incomplete fragments of conceptual schemas or concrete scenarios of use cases, example test programs are also reproduced after presenting the structural schema fragment.

During the definition and the execution of the presented CSTL tests we find errors and improvements to the osCommerce conceptual schema that we mark in blue color.

Some of the example test programs are inspired in real and live online stores based on *osCommerce*.

9.1. Main domain concepts

Figure 12 shows a simplified conceptual schema with the main domain concepts of the osCommerce system in order to provide a general overview of the system.

The products in the store are manufactured by **manufacturers**, are grouped into **categories** and belong to a **tax class**. Moreover, customers can write **reviews** of a product.

osCommerce is a multilingual system able to deal with any number of **languages**. Likewise, osCommerce allows working with different tax classes and **currencies**.

Products may have **attributes**. An attribute is an **option/value** pair which is used to offer multiple varieties of a product without needing to create many separate but very similar products. The price of a product is increased or decreased depending on the chosen attributes. The price variation produced by an attribute is indicated, for each product, by **product attribute** entity types.

Customers have one or more **addresses**. Each address is located in a **country**. If the country has **zones** (states or provinces) then the address must be located in one of its zones.

Every use of the online store is conceptually represented by a **session**. Sessions can be anonymous or belong to a customer. Moreover, every session has always a current currency and a current language.

In the context of sessions, users can surfing the online store. **Shopping carts** contain one or more selected items (not shown in the figure) each of which is a quantity of a product with a set of attributes.

When a customer confirms that he wants to buy the contents of his shopping cart the system generates an **order**. An order is made by a customer using a **payment method**. Furthermore, order prices are expressed in a specified **currency** and take into account the shipping costs, according to the chosen **shipping method.**

An order contains one or more **order lines**, each of which is a quantity of a product with a set of attributes.

Finally, osCommerce offers some administration tools like **banners**, used to customize the online advertisements in the store, and **newsletters**, used to send information by email to customers.

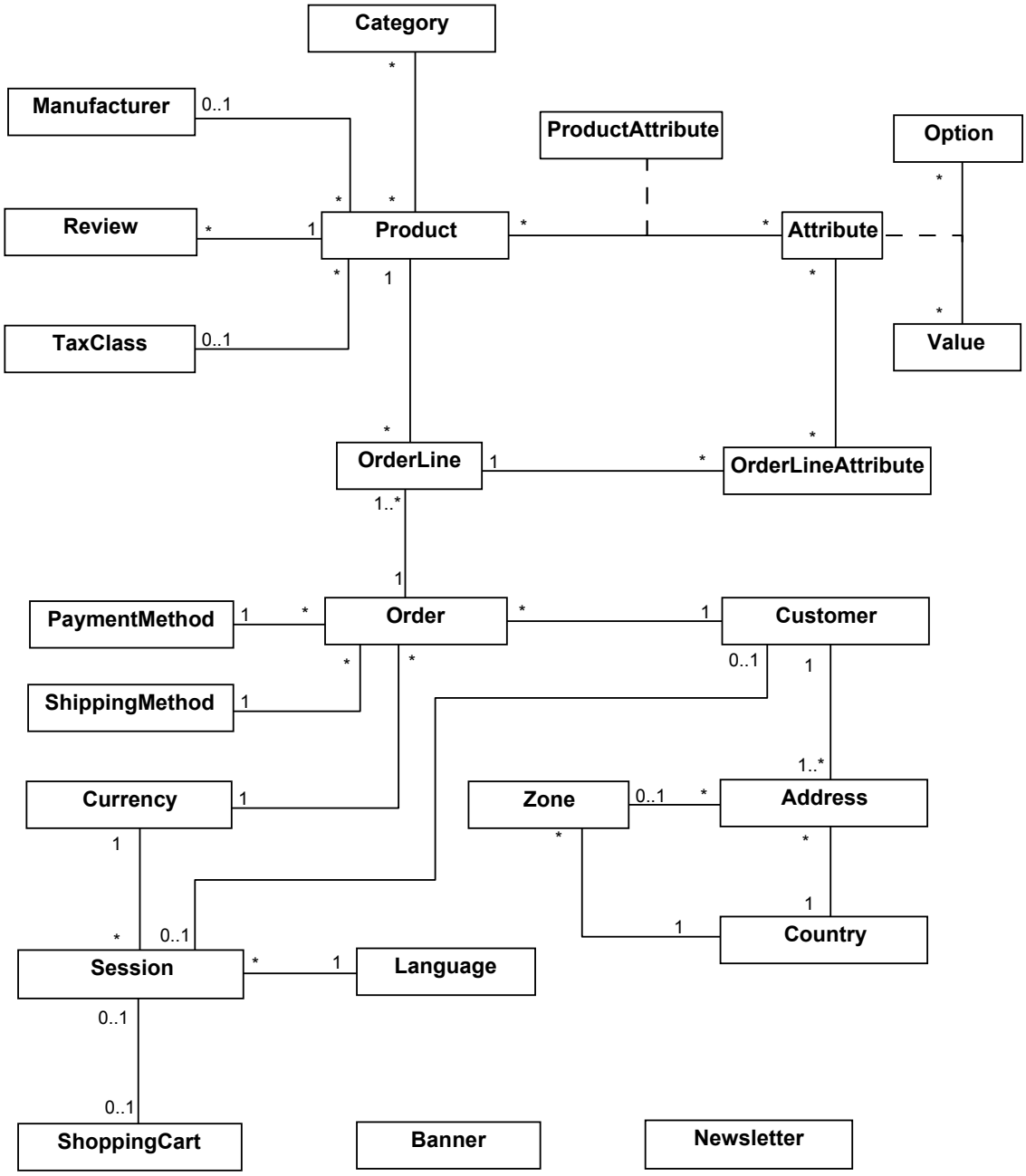
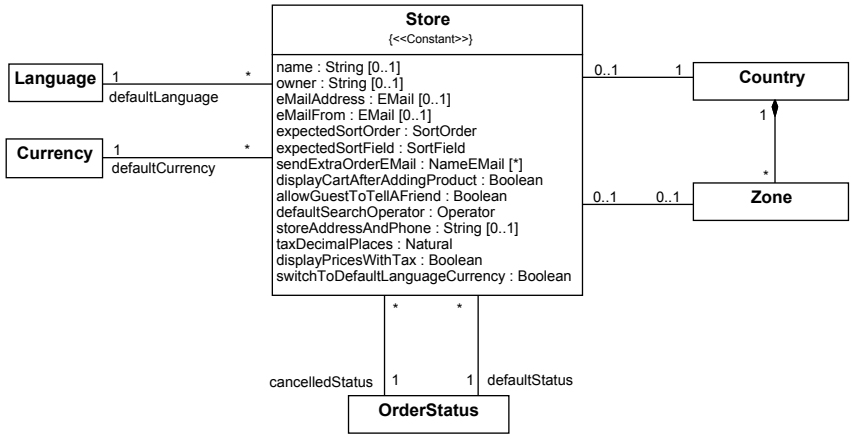


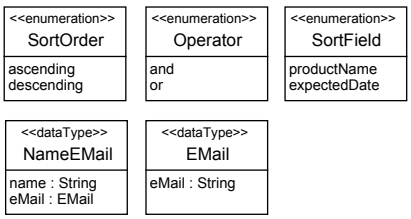
Fig. 12. Main domain concepts in the osCommerce Conceptual Schema

9.2. Store Data

Structural schema

osCommerce keeps general data about the store and some other information which is used to customize the behavior of the system.





[IC1] There is only one instance of *Store*

context Store::alwaysOneInstance: Boolean
body : Store.allInstances() -> size() = 1

[IC2] The store's zone is part of the country where the store is located.

context Store::zoneIsPartOfCountry: Boolean
body : self.zone -> notEmpty() **implies** self.country.zone -> includes (self.zone)

Example test program

```
testprogram InitializeStore{

        english:=new Language(name:='English', code:='EN');
        dollar:=new Currency(title:='USDollar', code:='USD');
        usa:=new Country(name:='United States', isoCode2:='US', isoCode3:='USA');
```

```
        spain:=new Country(name:='Spain', isoCode2:='ES', isoCode3:='ESP');
        newjersey := new Zone(name:='New Jersey', code:='NJ', country:=usa);
        catalonia := new Zone(name:='Catalonia', code:='CAT', country:=spain);

        cos:=new OrderStatus;
        cosl:=new OrderStatusInLanguage(language:=english,orderStatus:=cos);
        cosl.name:='cancelled';

        dos:=new OrderStatus;
        dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=english);
        dosl.name:='pending';

        test StoreInitializationWithDefaultMandatoryValues{
                s:=new Store(name:='JustArt');
                check inconsistency;

                s.defaultLanguage:=english;
                check inconsistency;

                s.defaultCurrency:=dollar;
                check inconsistency;

                s.country:=usa;
                check inconsistency;

                s.cancelledStatus:=cos;
                check inconsistency;

                s.defaultStatus:=dos;
                check consistency;
        }

        test OnlyOneStoreInstance{
                //We create the store 'JustArt'
                s:=new Store(name:='JustArt');
                s.defaultLanguage:=english;
                s.defaultCurrency:=dollar;
                s.country:=usa;
                s.cancelledStatus:=cos;
                s.defaultStatus:=dos;
                check consistency;

                //If we create another store, the state should be inconsistent
                s2:=new Store(name:='VirtualGallery');
                s2.defaultLanguage:=english;
                s2.defaultCurrency:=dollar;
                s2.country:=usa;
                s2.cancelledStatus:=cos;
                s2.defaultStatus:=dos;
                check inconsistency;
        }

        test StoreZoneMustBePartOfTheCountryWhereItIsLocated{
                //We create the store 'VirtualGallery'
                s:=new Store(name:='VirtualGallery');
                s.defaultLanguage:=english;
                s.defaultCurrency:=dollar;
                s.country:=usa;
                s.cancelledStatus:=cos;
                s.defaultStatus:=dos;
                check consistency;

                //We specify a zone which is not part of the USA
                s.zone := catalonia;
                check inconsistency;

                //We specify a correct zone
                s.zone := newjersey;
                check consistency;
        }
}
```

Use Cases

Change Store Data

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the initial values of the store data.

Main Success Scenario:

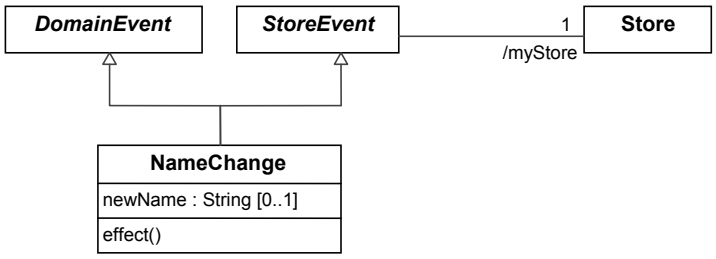
1. The system displays the current values of the store data.
2. The system administrator provides a new value for one of the store attributes:
   - [→*MameChange*]
   - [→*OwnerChange*]
   - [→*EMailAddressChange*]
   - [→*EMailFromChange*]
   - [→*ExpectedSortOrderChange*]
   - [→E*xpectedSortFieldChange*]
   - [→*SendExtraOrderChange*]
   - [→*DisplayCartAfterAddingProductChange*]
   - [→*AllowGuestToTellAFriendChange*]
   - [→*DefaultSearchOperatorChange*]
   - [→*StoreAddressAndPhoneChange*]
   - [→*TaxDecimalPlacesChange*]
   - [→*DisplayPricesWithTaxChange*]
   - [→S*witchToDefaultLanguageCurrencyChange*]
   - [→*CountryChange*]
   - [→*ZoneChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new values of the store data.

   The system administrator repeats steps 2-5 until he is done.

Note that if there are many similar events, we only reproduce the complete specification of the selected representative events used in the test program examples. The other events can be found in [41].

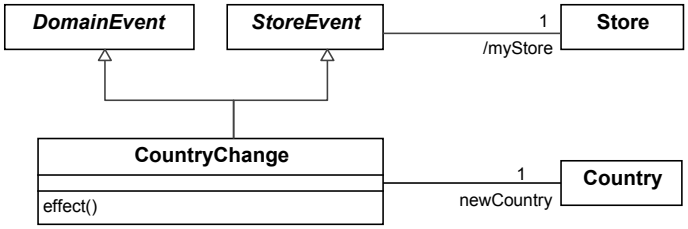
Events

NameChange



context NameChange::effect()
post : self.myStore.name = self.newName

CountryChange



context CountryChange::effect()
post : myStore.country = self.newCountry

Example test program

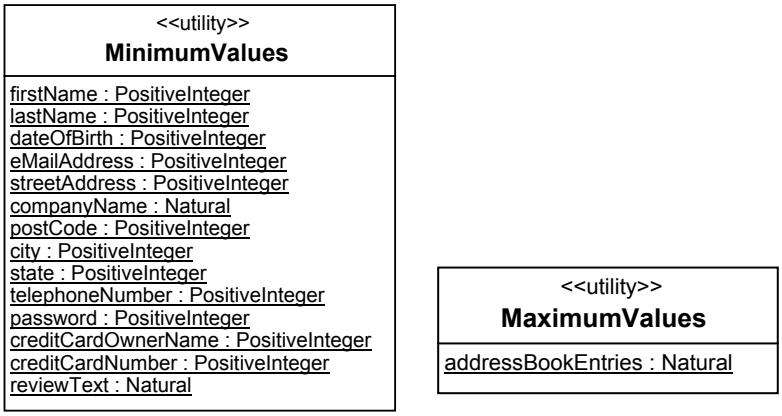
```
testprogram ChangeStoreData{

        //FIXTURE:InitializeStore
        s := new Store(name:='JustsArt');

        english := new Language(name:='English', code:='EN');
        s.defaultLanguage:=english;

        dollar := new Currency(title:='USDollar', code:='USD');
        s.defaultCurrency := dollar;


        spain := new Country
        (name:='Spain', isoCode2:='ES', isoCode3:='ESP');
        s.country:=spain;

        cos := new OrderStatus;
        cosl := new OrderStatusInLanguage(language:=english,orderStatus:=cos);
```

```
        cosl.name := 'cancelled';
        s.cancelledStatus := cos;

        dos := new OrderStatus;
        dosl := new OrderStatusInLanguage(orderStatus:=dos, language:=english);
        dosl.name:='pending';
        s.defaultStatus:=dos;

        //We test that name and country can be correctly changed.
        test NameAndCountryChange{
                assert equals s.name 'JustsArt';
                new NameChange(newName:='JustArt') occurs;
                assert equals s.name 'JustArt';

                assert equals s.country spain;
                usa := new Country
                   (name:='United States', isoCode2:='US', isoCode3:='USA');
                new CountryChange(newCountry:=usa);
                assert equals s.country usa;
        }
}
```

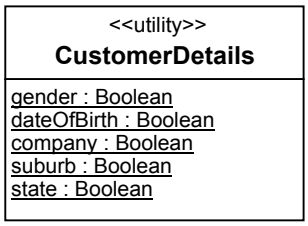
9.3. Configuration values

Structural schema

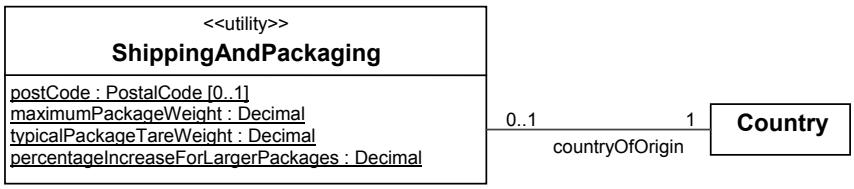
osCommerce allows defining and changing the minimum and maximum length for some *String* attributes related to customer details.

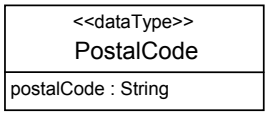


The system also allows specifying whether some customer attributes are shown and required when creating, editing or showing an account.



The system allows setting up some configuration values used in shipping costs calculation.

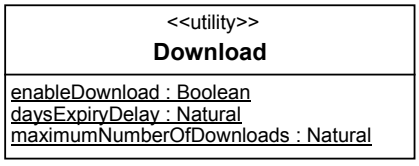




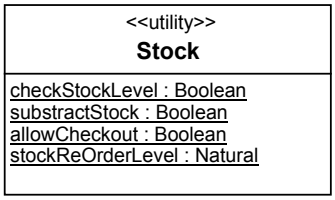
[IC1] The package tare weight must be less than the maximum package weight.

context ShippingAndPackaging::tareIsLessThanMaximumWeight: Boolean
body : self.typicalPackageTareWeight < self.maximumPackageWeight

The system allows customizing the most important general downloadable product properties.



The system allows configuring some options about the stock administration.



Use Cases

Assign minimum values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the minimum values of some attributes.

Main Success Scenario:

1. The system displays the current minimum values.
2. The system administrator provides a new value for one of the minimum values:
   - [→*FirstNameMinimumChange*]
   - [→*LastNameMinimumChange*]
   - [→*DateOfBirthMinimumChange*]
   - [→*EMailAddressMinimumChange*]
   - [→*StreetAddressMinimumChange*]
   - [→*CompanyNameMinimumChange*]
   - [→*PostCodeMinimumChange*]
   - [→*CityMinimumChange*]
   - [→*StateMinimumChange*]
   - [→*TelephoneMinimumChange*]
   - [→*PasswordMinimumChange*]
   - [→*CreditCardOwnerNameMinimumChange*]
   - [→*CreditCardNumberMinimumChange*]
   - [→*ReviewTextMinimumChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current minimum values.

   The system administrator repeats steps 2-5 until he is done.

Assign maximum values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the maximum number of address book entries permitted for each customer.

Main Success Scenario:

1. The system displays the current maximum number of address book entries for each customer.
2. The system administrator provides the new maximum value:
   [→*AddressBookEntriesMaximumChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current maximum value.

Change shown customer details

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change whether some customer attributes are shown.

Main Success Scenario:

1. The system displays the current values of customer details configuration (shown or not shown).
2. The system administrator provides the new value for one of the customer details:
   [→*GenderCustomerDetailChange*]
   [→*DateOfBirthCustomerDetailChange*]
   [→*CompanyCustomerDetailChange*]
   [→*SuburbCustomerDetailChange*]
   [→*StateCustomerDetailChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current values of customer details configuration.

   The system administrator repeats steps 2-5 until he is done.

Assign shipping and packaging configuration values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the shipping and packaging configuration values.

Main Success Scenario:

1. The system displays the current shipping and packaging configuration values.
2. The system administrator provides the new value for one of the shipping and packaging configurable options:
   [→*PostCodeShippingConfigurationChange*]
   [→*MaximumPackageWeightShippingConfigurationChange*]
   [→*TypicalPackageTareWeightShippingConfigurationChange*]
   [→*PercentageIncreaseForLargerPackagesShippingConfigurationChange*]
   [→*CountryShippingConfigurationChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current shipping and packaging configuration values.

   The system administrator repeats steps 2-5 until he is done.

Change download configuration values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the download configuration values.

Main Success Scenario:

1. The system displays the current download configuration values.
2. The system administrator provides the new value for one of the download configuration options:
   [→*EnableDownloadConfigurationChange*]
   [→*DaysExpiryDelayDownloadConfigurationChange*]
   [→*MaximumNumberDownloadConfigurationChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current download configuration values.

   The system administrator repeats steps 2-5 until he is done.

Change stock configuration values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the stock configuration values.

Main Success Scenario:

1. The system displays the current stock configuration values.
2. The system administrator provides the new value for one of the stock configuration options:

   [→*CheckLevelStockConfigurationChange*]

   [→*SubstractStockConfigurationChange*]

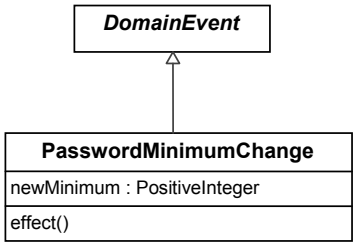
   [→*AllowCheckoutStockConfigurationChange*]

   [→*ReorderLevelStockConfigurationChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current stock configuration values.

   The system administrator repeats steps 2-5 until he is done.

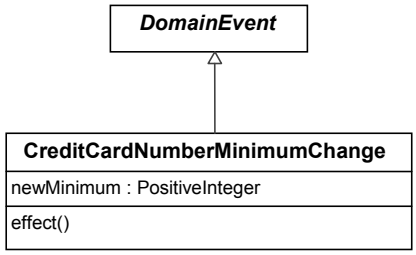
Events

PasswordMinimumChange



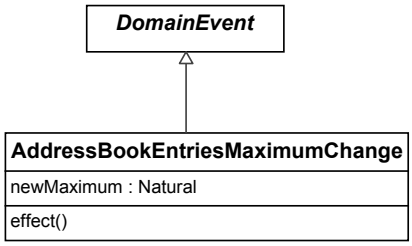
context PasswordMinimumChange::effect()
post : MinimumValues.password = self.newMinimum

CreditCardNumberMinimumChange



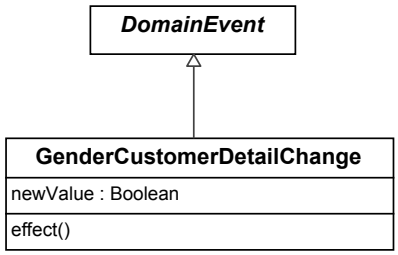
context CreditCardNumberMinimumChange::effect()
post : MinimumValues.creditCardNumber = self.newMinimum

AddressBookEntriesMaximumChange



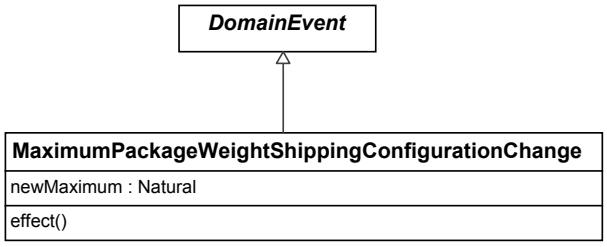
context AddressBookEntriesMaximumChange::effect()
post : MaximumValues.addressBookEntries = self.newMaximum

GenderCustomerDetailChange



context GenderCustomerDetailChange::effect()
post : CustomerDetails.gender = self.newValue

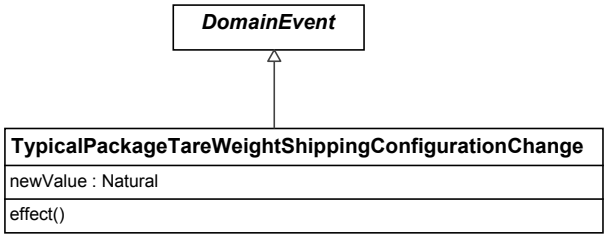
MaximumPackageWeightShippingConfigurationChange



«IniIC»
context MaximumPackageWeightShippingConfigurationChange::maxIsGreaterThanTypicalWeight():Boolean
body : self.newMaximum > ShippingAndPackaging.typicalPackageTareWeight

context MaximumPackageWeightShippingConfigurationChange::effect()
post : ShippingAndPackaging.maximumPackageWeight = self.newMaximum

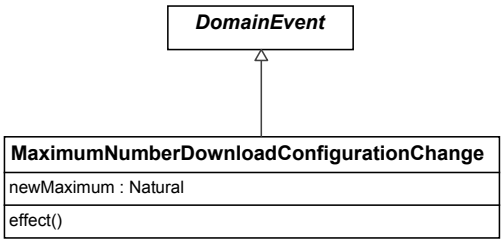
TypicalPackageTareWeightShippingConfigurationChange



context TypicalPackageTareWeightShippingConfigurationChange::effect()
post : ShippingAndPackaging.typicalPackageTareWeight = self.newValue

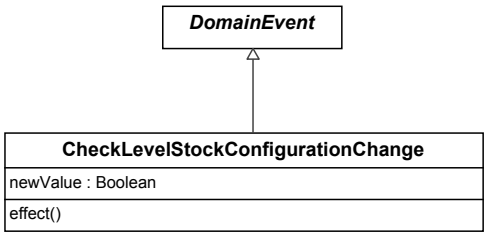
«IniIC»
context TypicalPackageTareWeightShippingConfigurationChange::valueDoesNotExceedMaxWeight():Boolean
body : self.newValue < ShippingAndPackaging.maximumPackageWeight

MaximumNumberDownloadConfigurationChange



context MaximumNumberDownloadConfigurationChange::effect()
post : Download.maximumNumberOfDownloads= self.newMaximum

CheckLevelStockConfigurationChange



context CheckLevelStockConfigurationChange::effect()
post : Stock.checkStockLevel= self.newValue

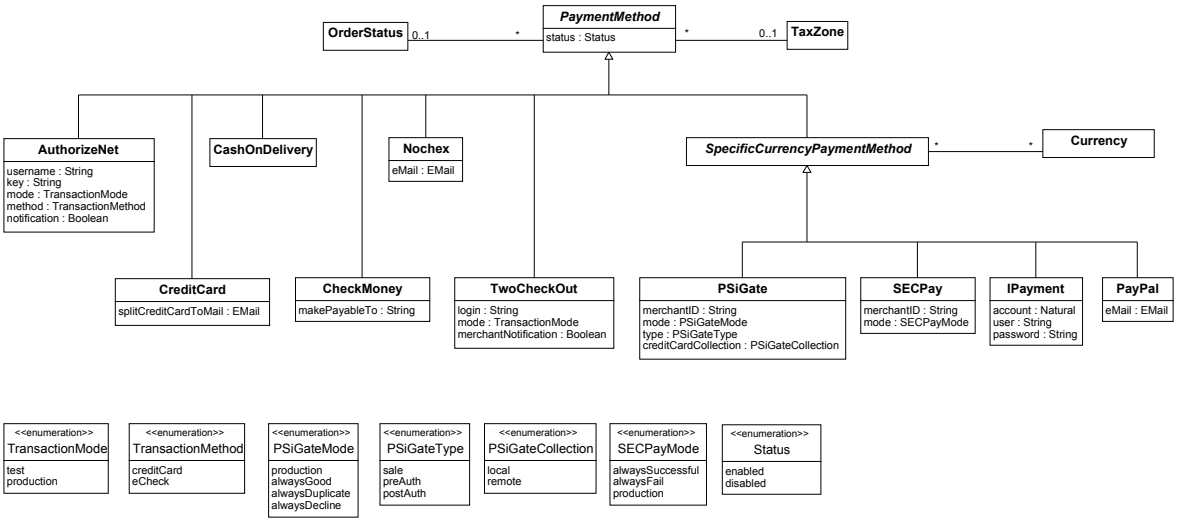
Example test program

```
testprogram ConfigurationValues{

        //We create an instance of the entity types
        //MaximumValues and MinimumValues (multiple classification)

        configurationValues := new MaximumValues, MinimumValues,
                               CustomerDetails, ShippingAndPackaging, Download, Stock;
        spain:=new Country(name:='Spain', isoCode2:='ES', isoCode3:='ESP');
        configurationValues.countryOfOrigin := spain;
        configurationValues.maximumPackageWeight := 30;
        configurationValues.typicalPackageTareWeight := 15;

        test ChangeMinimumAndMaximumValues{
                //The postconditions of the following events are automatically checked
                new PasswordMinimumChange(newMinimum:=8) occurs;
                new CreditCardNumberMinimumChange(newMinimum:=16) occurs;
                new AddressBookEntriesMaximumChange(newMaximum:=3) occurs;
                new GenderCustomerDetailChange(newValue:=true) occurs;
                new MaximumNumberDownloadConfigurationChange(newMaximum:=5) occurs;
                new CheckLevelStockConfigurationChange(newValue:=false) occurs;
                new TypicalPackageTareWeightShippingConfigurationChange(newValue:=10) occurs;
                new MaximumPackageWeightShippingConfigurationChange(newMaximum:=25) occurs;
        }

        test InconsistentShippingConfiguration{
                //The typical package weight cannot be greater than the maximum package weight
                new TypicalPackageTareWeightShippingConfigurationChange
                                                        (newValue:=40) may not occur;
                new MaximumPackageWeightShippingConfigurationChange
                                                        (newMaximum:=10) may not occur;
        }
}
```

9.4. Payment methods

Structural schema

The system allows operating with different payment methods.



[IC1] There is at least one enabled payment method

```
context PaymentMethod::atLeastOneEnabled: Boolean
  body : PaymentMethod.allInstances() -> select (pm | pm.status=Status::enabled) -> size() >= 1
```

Use Cases

Install a payment method

Primary Actor: Store administrator

Precondition: The payment method is not installed yet.

Trigger: The store administrator wants to install a payment method.

Main Success Scenario:

1. The system shows all the available payment methods and which of they are installed.
2. The store administrator selects a non installed payment method.
3. The store administrator provides the data of the payment method:
   - [→*InstallAuthorizeNetPaymentMethod*]
   - [→*InstallCreditCardPaymentMethod*]
   - [→*InstallCashOnDeliveryPaymentMethod*]
   - [→*InstallIPaymentPaymentMethod*]
   - [→*InstallCheckMoneyPaymentMethod*]
   - [→*InstallNochexPaymentMethod*]
   - [→*InstallPayPalPaymentMethod*]
   - [→*InstallTwoCheckOutPaymentMethod*]
   - [→*InstallPSiGatePaymentMethod*]
   - [→*InstallSECPaymentMethod*]
4. The system validates that the data is correct.
5. The system uninstalls the new payment method and enables it.

Uninstall a payment method

Primary Actor: Store administrator

Precondition: The payment method is installed and there is at least another payment method enabled.

Trigger: The store administrator wants to uninstall a payment method.

Main Success Scenario:

1. The system shows all the payment methods and which of they are installed.
2. The store administrator selects an installed payment method.
   [→*UninstallAuthorizeNetPaymentMethod*]
   [→*UninstallCreditCardPaymentMethod*]
   [→*UninstallCashOnDeliveryPaymentMethod*]
   [→*UninstallIPaymentPaymentMethod*]
   [→*UninstallCheckMoneyPaymentMethod*]
   [→*UninstallNochexPaymentMethod*]
   [→*UninstallPayPalPaymentMethod*]
   [→*UninstallTwoCheckOutPaymentMethod*]
   [→*UninstallPSiGatePaymentMethod*]
   [→*UninstallSECPaymentMethod*]
3. The system uninstalls the selected payment method.

Extensions:

2a. The payment method is used in an existing order:

2a1. The system warns the store administrator that the payment method is used in the information of existing orders and that is only possible to disable the payment method.

2a2. The system changes the status of the payment method to disabled.

[→*StatusPaymentMethodChange*]

2a3. The use case ends.

Change payment method values

Primary Actor: System administrator

Precondition: The payment method is installed.

Trigger: The system administrator wants to change the configuration values of an installed payment method.

Main Success Scenario:

1. The system displays the installed payment methods.
2. The customer selects an installed payment method.
3. The system displays the current values of the payment method.
4. The system administrator provides the new values for the configurable attributes of the payment method:
   [→*EditAuthorizeNetPaymentMethod*]
   [→*EditCreditCardPaymentMethod*]
   [→*EditCashOnDeliveryPaymentMethod*]

[→*EditIPaymentPaymentMethod*]

[→*EditCheckMoneyPaymentMethod*]

[→*EditNochexPaymentMethod*]

[→*EditPayPalPaymentMethod*]

[→*EditTwoCheckOutPaymentMethod*]

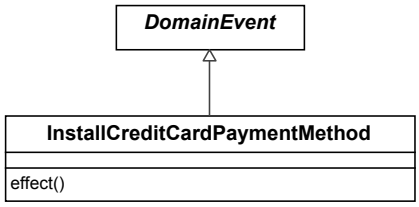
[→*EditPSiGatePaymentMethod*]

[→*EditSECPaymentMethod*]

5. The system validates that the new values are correct.
6. The system saves the new values.
7. The system displays the new values of the payment method.

Events

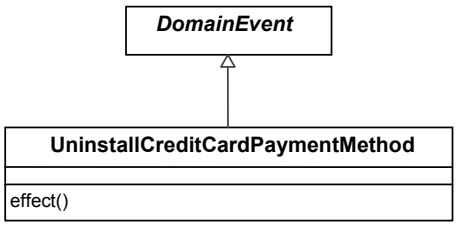
InstallCreditCardPaymentMethod



«IniIC»
context InstallCreditCardPaymentMethod::paymentMethodIsNotInstalled():Boolean
body : CreditCard.allInstances() -> isEmpty()

context InstallCreditCardPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(CreditCard) **and** pm.status=Status::enabled

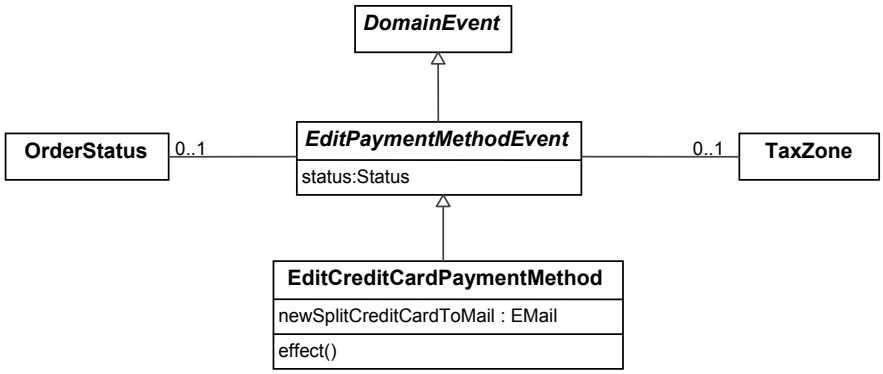
UninstallCreditCardPaymentMethod



«IniIC»
context UninstallCreditCardPaymentMethod::paymentMethodCanBeUninstalled():Boolean
body : CreditCard.allInstances() -> notEmpty() **and**
(PaymentMethod.allInstances-Set{CreditCard.allInstances->any(true)})->exists(pm|pm.status=#enabled)

context UninstallCreditCardPaymentMethod::effect()
post : CreditCard.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

EditCreditCardPaymentMethod



«IniIC»
context EditCreditCardPaymentMethod::paymentMethodIsInstalled():Boolean
body : CreditCard.allInstances() -> notEmpty()

«IniIC»
context EditCreditCardPaymentMethod::atLeastOneEnabled():Boolean
body :
self.status=Status::disabled
implies
(PaymentMethod.allInstances-Set{CreditCard.allInstances->any(true)})
->exists(pm | pm.status=Status::enabled)

context EditCreditCardPaymentMethod::effect()
post :
let pm:CreditCard **=** CreditCard.allInstances() -> any(true) **in**
pm.splitCreditCardToMail=self.newSplitCreditcardToMail **and**
pm.status=self.status **and**
pm.orderStatus=self.orderStatus **and** pm.taxZone=self.taxZone

Example test program

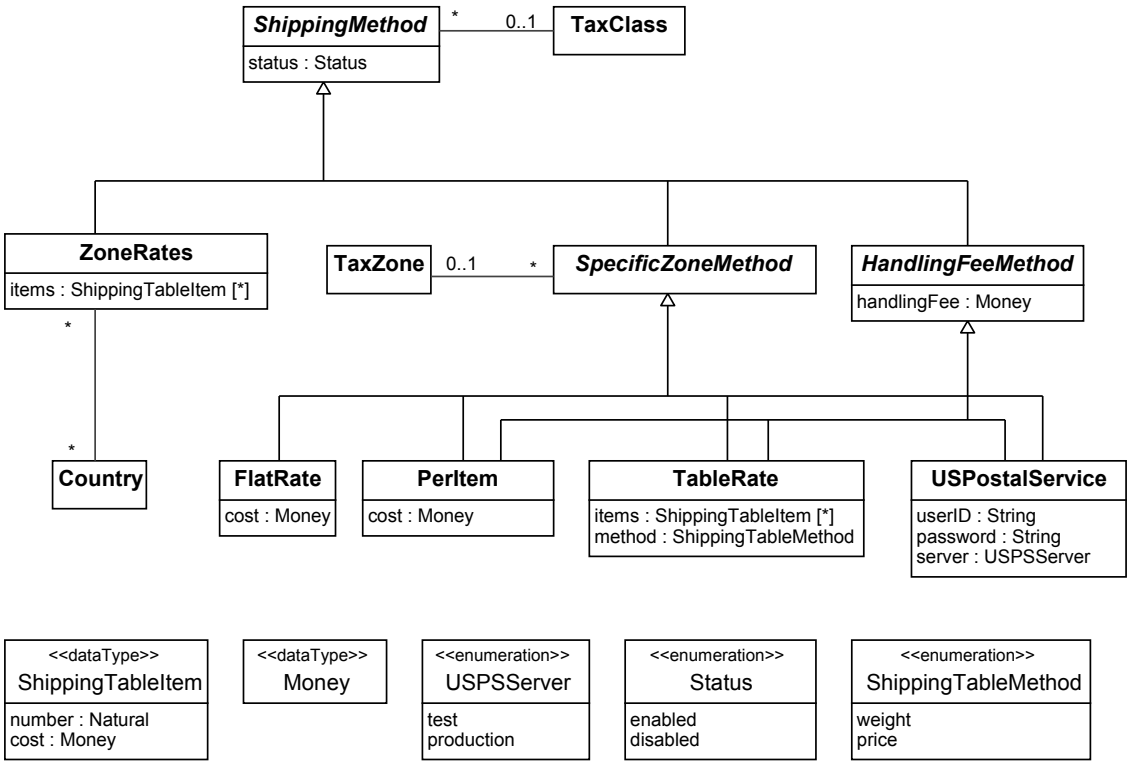
```
testprogram InstallUninstallAndEditPaymentMethods{

        test InstallCreditCardOnce{
                new InstallCreditCardPaymentMethod occurs;
        }

        test InstallCreditCardTwice{
                new InstallCreditCardPaymentMethod occurs;
                new InstallCreditCardPaymentMethod may not occur;
        }

        test UninstallCreditCardAlreadyInstalled{
           new InstallCreditCardPaymentMethod occurs;
           //We cannot uninstall the credit card method because
           //there is no other payment method enabled
           new UninstallCreditCardPaymentMethod may not occur;
           new InstallCashOnDeliveryPaymentMethod occurs;
           new UninstallCreditCardPaymentMethod occurs;
        }

        test AtLeastOnePaymentMethodEnabled{
                new InstallCreditCardPaymentMethod occurs;
```

```
            //We cannot disable the credit card method because
            //there is no other payment method enabled
            new EditCreditCardPaymentMethod(status:=#disabled) may not occur;
            new InstallCashOnDeliveryPaymentMethod occurs;
            new EditCreditCardPaymentMethod(status:=#disabled) occurs;
        }

        test UninstallCreditCardNotInstalledYet{
           new UninstallCreditCardPaymentMethod may not occur;
        }
```

9.5. Shipping methods

Structural schema

The system allows operating with different shipping methods.



[IC1] There is at least one enabled shipping method.

```
context ShippingMethod::atLeastOneEnabled: Boolean
 body : ShippingMethod.allInstances() -> select (sm | sm.status=Status::enabled) -> size() >= 1
```

Use Cases

Install a shipping method

Primary Actor: Store administrator

Precondition: The shipping method is not installed yet.

Trigger: The store administrator wants to install a shipping method.

Main Success Scenario:

1. The system shows all the available shipping methods and which of they are installed.
2. The store administrator selects a non installed shipping method.
3. The store administrator provides the data of the shipping method.
   [→*InstallZoneRatesShippingMethod*]
   [→*InstallFlatRateShippingMethod*]
   [→*InstallPerItemShippingMethod*]
   [→*InstallTableRateShippingMethod*]
   [→*InstallUSPostalServiceShippingMethod*]
4. The system validates that the data is correct.
5. The system creates an instance of the new shipping method and enables it.

Uninstall a shipping method

Primary Actor: Store administrator

Precondition: The shipping method is installed and there is at least another shipping method enabled.

Trigger: The store administrator wants to uninstall a shipping method.

Main Success Scenario:

1. The system shows all the available shipping methods and which of they are installed.
2. The store administrator selects an installed shipping method.
   [→*UninstallZoneRatesShippingMethod*]
   [→*UnistallFlatRateShippingMethod*]
   [→*UninstallPerItemShippingMethod*]
   [→*UninstallTableRateShippingMethod*]
   [→*UninstallUSPostalServiceShippingMethod*]
3. The system deletes the instance of the selected shipping method.

Extensions:

2a. The shipping method is the shipping method used in an existing order:

2a1. The system warns the store administrator that the shipping method is used in the information of existing orders and that is only possible to disable the shipping method.

2a2. The system changes the *enabled* attribute of the shipping method to false:

[→*StatusShippingMethodChange*]

2a3. The use case ends.

Change shipping method values

Primary Actor: System administrator

Precondition: The shipping method is installed.

Trigger: The system administrator wants to change the configuration values of an installed shipping method.

Main Success Scenario:

1. The system displays the installed shipping methods.
2. The customer selects an installed shipping method.
3. The system displays the current values of the selected shipping method.
4. The system administrator provides the new values for the configurable attributes of the shipping method:

   [→*EditZoneRatesShippingMethod*]

   [→*EditFlatRateShippingMethod*]

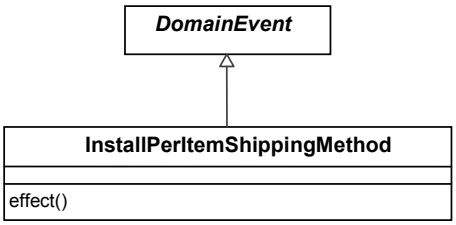
   [→*EditPerItemShippingMethod*]

   [→*EditTableRateShippingMethod*]

   [→*EditUSPostalServiceShippingMethod*]
5. The system validates that the new values are correct.
6. The system saves the new values.
7. The system displays the new values of the shipping method.

Events

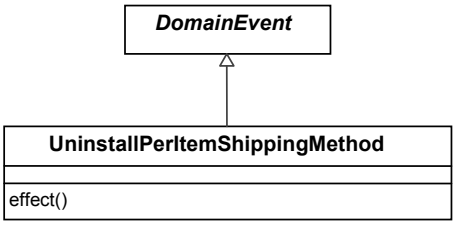
InstallPerItemShippingMethod



«IniIC»
context InstallPerItemShippingMethod::ShippingMethodIsNotInstalled():Boolean
body : PerItem.allInstances() -> isEmpty()

context InstallPerItemShippingMethod::effect()
post : sm.oclIsNew() **and** sm.oclIsTypeOf(PerItem) **and** sm.status=Status::enabled

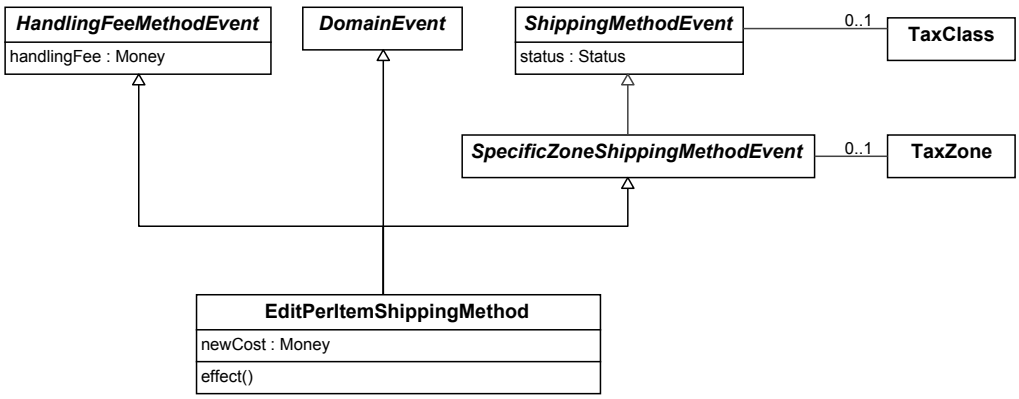
UninstallPerItemShippingMethod



«IniIC»
context UninstallPerItemShippingMethod::ShippingMethodCanBeUninstalled():Boolean
body :
PerItem.allInstances() -> notEmpty() **and**
(ShippingMethod.allInstances-Set{PerItem.allInstances->any(true)})->exists(sm|sm.status=#enabled)

context UninstallPerItemShippingMethod::effect()
post : PerItem.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

EditPerItemShippingMethod



«IniIC»
context EditPerItemShippingMethod::paymentMethodIsInstalled():Boolean
body : PerItem.allInstances() -> notEmpty()

«IniIC»
context EditPerItemShippingMethod::atLeastOneEnabled:Boolean
body:
self.status=Status::disabled
implies
(ShippingMethod.allInstances-Set{PerItem.allInstances->any(true)})
->exists(pm | pm.status=Status::enabled)

context EditPerItemShippingMethod::effect()
post :
let sm: PerItem= PerItem.allInstances() -> any(true) **in**
sm.cost=self.newCost **and**
sm.handlingFee=self.handlingFee **and**
sm.taxZone=self.taxZone **and**
sm.taxClass=self.taxClass **and**
sm.status = self.status

Example test program

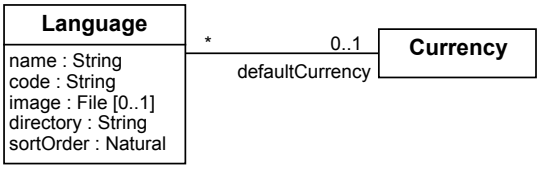
```
testprogram InstallUninstallShippingMethods{

        test InstallPerItemShippingMethodOnce{
                new InstallPerItemShippingMethod occurs;
        }

        test InstallPerItemShippingMethodTwice{
                new InstallPerItemShippingMethod occurs;
                new InstallPerItemShippingMethod may not occur;
        }

        test UninstallPerItemShippingMethodAlreadyInstalled{
           new InstallPerItemShippingMethod occurs;
```

```
            //We cannot uninstall PerItem method because there is no
            //other shipping method enabled
            new InstallFlatRateShippingMethod occurs;
            new UninstallPerItemShippingMethod occurs;
        }
        test UninstallCreditCardNotInstalledYet{
            new UninstallPerItemShippingMethod may not occur;
        }

        test AtLeastOneShippingMethodEnabled{
                new InstallPerItemShippingMethod occurs;
                new EditPerItemShippingMethod(status:=#disabled) may not occur;
                //Only if there is another shipping method enabled,
                //we can change PerItem to disabled
                new InstallFlatRateShippingMethod occurs;
                new EditPerItemShippingMethod(status:=#disabled) occurs;
        }
}
```

9.6. Languages

Structural schema

osCommerce is a multilingual system able to deal with any number of languages.



[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)

Use Cases

Add a language

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new language.

Main Success Scenario:

1. The store administrator provides the details of the new language:

   [→*NewLanguage*]

2. The system validates that the data is correct.
3. The system saves the new language.

Edit a language

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a language.

Main Success Scenario:

1. The store administrator selects the language to be edited.
2. The store administrator provides the new details of the selected language:

   [→*EditLanguage*]

3. The system validates that the data is correct.
4. The system saves the changes.

Delete a language

Primary Actor: Store administrator

Precondition: There are at least two languages.

Trigger: The store administrator wants to delete a language.

Main Success Scenario:

1. The store administrator selects the language to be deleted.
2. The store administrator confirms that he wants to delete the language:

   [→*DeleteLanguage*]

3. The system deletes the language.

Extensions:

2a. The deleted language is the default language of the store.

   2a1. The system sets any of the available languages as the default language:

   [→*SetDefaultLanguage*]

2b. The deleted language is the current language of any active session.

   2b1. The system sets any of the available languages as the current language:

   [→*SetCurrentLanguage*]

Set the default language

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the default language.

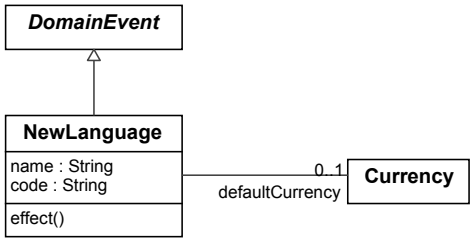
Main Success Scenario:

1. The store administrator selects the language which will become the default language.
2. The system updates the default language:

   [→*SetDefaultLanguage*]

Events

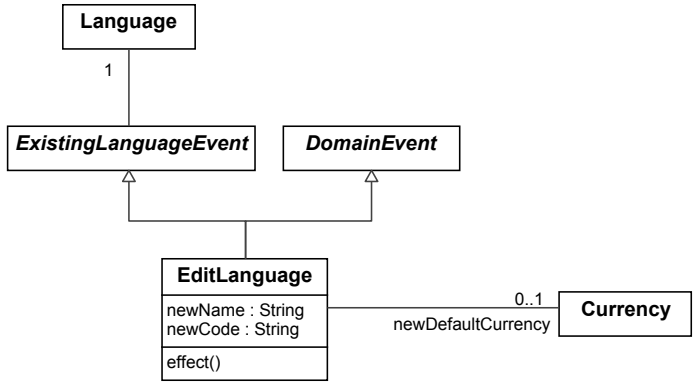
NewLanguage



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

```
context  NewLanguage::effect()
  post :
     l.oclIsNew()  and
     l.oclIsTypeOf(Language) and
     l.name = self.name and
     l.code = self.code and
     l.defaultCurrency = self.defaultCurrency
```

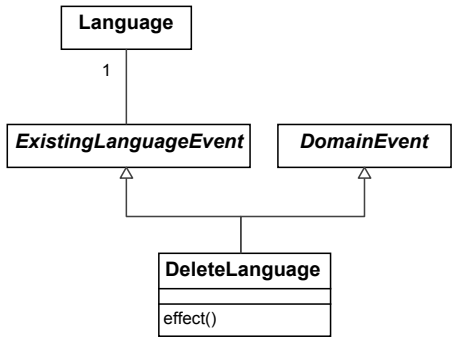
EditLanguage



«IniIC»
context EditLanguage::languageDoesNotExist(): Boolean
body:
not ((Language.allInstances-Set{self.language})
->exists(name=self.newName or code=self.newCode))

context EditLanguage::effect()
post :
self.language.name = self.newName **and**
self.language.code = self.newCode **and**
self.language.defaultCurrency = self.newDefaultCurrency

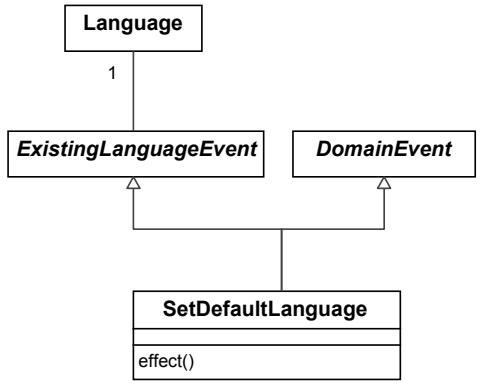
DeleteLanguage



«IniIC»
context DeleteLanguage::AtLeastTwoLanguages(): Boolean
body : Language.allInstances() -> size() >= 2

context DeleteLanguage::effect()
post: **not** self.language@pre.oclIsKindOf(OclAny)

SetDefaultLanguage



context SetDefaultLanguage::effect()
post : Store.allInstances() -> any(true).defaultLanguage = self.language

Example test program

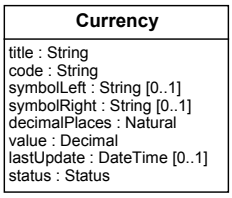
```
testprogram LanguageManagement{

        dollar:=new Currency(title:='USDollar', code:='USD');

        test InstallLanguage{
                 new NewLanguage(newName:='English', newCode:='EN') occurs;
        }

        test InstallLanguagesTwice{
                new NewLanguage(newName:='English', newCode:='EN') occurs;
                new NewLanguage(newName:='English', newCode:='EN') may not occur;
        }

        test InstallLanguageWithDefaultCurrency{
                new NewLanguage(newName:='English', newCode:='EN', defaultCurrency:=dollar)
                occurs;
        }

        test EditLanguage{
                new NewLanguage(newName:='Englishhh', newCode:='EN') occurs;
                createdLanguage:=Language.allInstances->select(name='Englishhh')->any(true);
                new EditLanguage
                   (language:=createdLanguage, newName:='English', newCode:='EN') occurs;
                assert equals l.name 'English';

                //We cannot edit a language if it causes duplicated languages
                catalan := new Language(name:='Catalan', code:='CAT');
                new EditLanguage(language:=createdLanguage,newName:='Catalan', newCode:='EN')
                may not occur;
        }

        test DeleteLanguage{
                //We cannot delete a language if there are no other languages enabled
                english := new Language(name:='English', code:='EN', defaultCurrency:=dollar);
                new DeleteLanguage(language:=english) may not occur;
                catalan := new Language(name:='Catalan', code:='CAT');
                new DeleteLanguage(language:=english) occurs;
        }
```

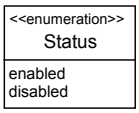
```
        test SetDefaultLanguage{
                //Initialize store
                english:=new Language(name:='English', code:='EN');
                usa:=new Country(name:='United States', isoCode2:='US', isoCode3:='USA');
                cos:=new OrderStatus;
                cosl:=new OrderStatusInLanguage(language:=english,orderStatus:=cos);
                cosl.name:='cancelled';
                dos:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=english);
                dosl.name:='pending';
                s:=new Store(name:='VirtualGallery');
                s.defaultCurrency:=dollar;
                s.country:=usa;
                s.cancelledStatus:=cos;
                s.defaultStatus:=dos;
                s.defaultLanguage:=english;

                //We test that a new language is set as default language
                spanish:=new Language(name:='Spanish', code:='ESP');
                new SetDefaultLanguage(language:=spanish)occurs;
                assert equals s.defaultLanguage spanish;
                assert not equals s.defaultLanguage english;
        }
}
```

9.7. Currencies

Structural schema

osCommerce allows working with different currencies.





[IC1] A currency is identified by its title and by its code.

context Currency::codeAndTitleAreUnique: Boolean
body :
Currency.allInstances() -> isUnique(title) **and**
Currency.allInstances() -> isUnique(code)

Use Cases

Add a currency

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new currency.

Main Success Scenario:

1. The store administrator provides the details of the new currency:

   [→*NewCurrency*]
2. The system validates that the data is correct.
3. The system saves the new currency and enables it.

Edit a currency

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a currency.

Main Success Scenario:

1. The store administrator selects the currency to be edited.
2. The store administrator provides the new details of the selected currency:

   [→*EditCurrency*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a currency

Primary Actor: Store administrator

Precondition: There is at least another enabled currency.

Trigger: The store administrator wants to delete a currency.

Main Success Scenario:

1. The store administrator selects the currency to be deleted.

2. The store administrator confirms that he wants to delete the currency:

   [→*DeleteCurrency*]

3. The system deletes the currency.

Extensions:

2a. The deleted currency was the default currency.

   2a1. The system sets any of the available currencies as the default currency:

      [→*SetDefaultCurrency*]

2b. The deleted currency is the current currency of an active session.

   2b1. The system sets any of the available currencies as the current currency:

      [→*SetCurrentCurrency*]

2c. The currency is the currency of an order:

   2c1. The system changes the status of the currency to disable.

      [→*CurrencyStatusChange*]

   2c2. The use case ends.

Update currencies

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to update automatically via Internet the change values for currencies.

Main Success Scenario:

1. The system connects to the change information server.
2. The value change is automatically updated for all the currencies:

   [→*UpdateCurrencyValueChange*]

Set the default currency

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the default currency.

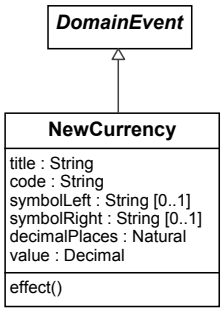
Main Success Scenario:

1. The store administrator selects the currency which will become the default currency.
2. The system updates the default currency:

   [→*SetDefaultCurrency*]

Events

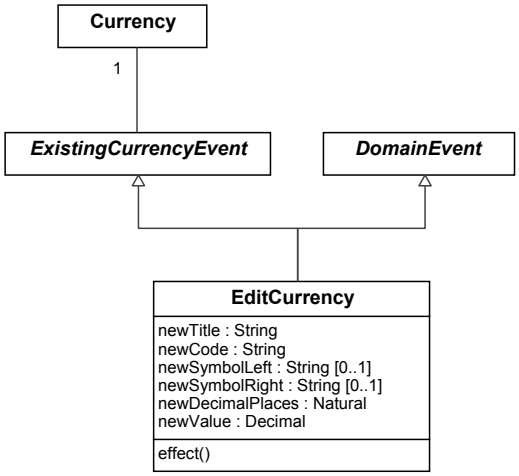
NewCurrency



«IniIC»
context NewCurrency::currencyDoesNotExist(): Boolean
body :
not Currency.allInstances() -> exists(c | c.title=self.title **and**
c.code=self.code)

context NewCurrency::effect()
post :
c.oclIsNew() **and**
c.oclIsTypeOf(Currency) **and**
c.title = self.title **and**
c.code = self.code **and**
c.symbolLeft = self.symbolLeft **and**
c.symbolRight = self.symbolRight **and**
c.decimalPlaces = self.decimalPlaces **and**
c.value = self.value **and**
c.status = Status::enabled

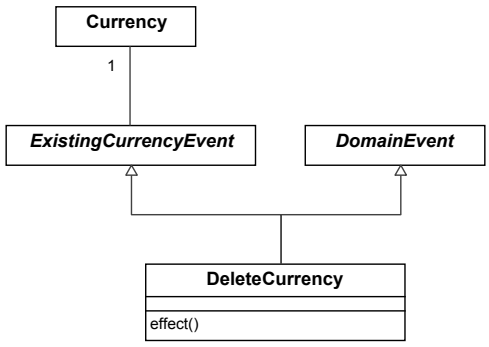
EditCurrency



«IniIC»
context EditCurrency::currencyDoesNotExist(): Boolean
  body:
    not ((Currency.allInstances-Set{self.currency})->exists(title=self.newTitle or code=self.newCode))

context EditCurrency::effect()
  post :
    currency.title = self.newTitle **and**
    currency.code = self.newCode **and**
    currency.symbolLeft = self.newSymbolLeft **and**
    currency.symbolRight = self.newSymbolRight **and**
    currency.decimalPlaces = self.newDecimalPlaces **and**
    currency.value = self.newValue

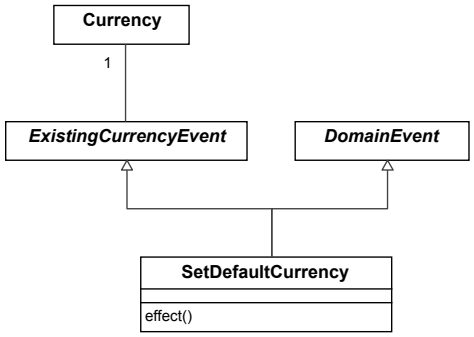
DeleteCurrency



«IniIC»
context DeleteCurrency::AtLeastTwoCurrencies(): Boolean
  body : Currency.allInstances() -> size() >= 2

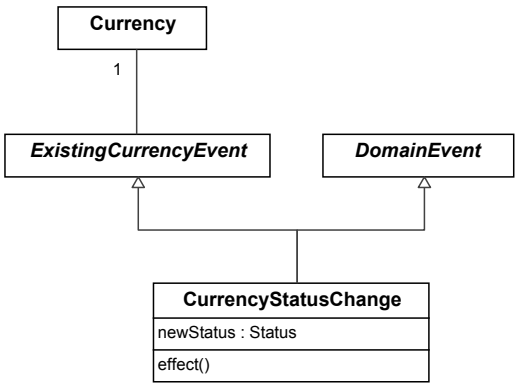
context DeleteCurrency::effect()
  post: **not** self.currency@pre.oclIsKindOf(OclAny)

SetDefaultCurrency



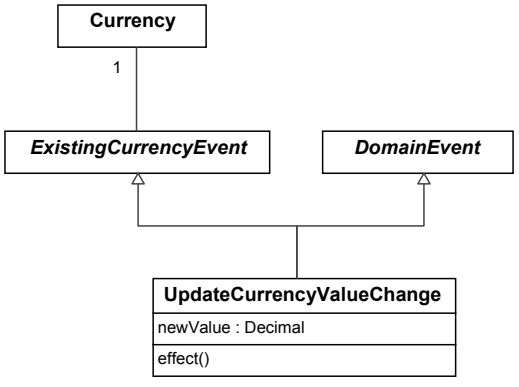
```
context SetDefaultCurrency::effect()
  post : Store.allInstances() -> any(true).defaultCurrency = self.currency
```

CurrencyStatusChange




```
«IniIC»
context CurrencyStatusChange::atLeastOneCurrencyEnabled():Boolean
   body:
      self.newStatus=Status::disabled
      implies
      (Currency.allInstances-Set{self.currency})->exists(c|c.status=Status::enabled)

context CurrencyStatusChange::effect()
  post : self.currency.status = self.newStatus
```

UpdateCurrencyValueChange




```
context UpdateCurrencyValueChange::effect()
  post : self.currency.value = self.newValue
  post : self.currency.lastUpdated = Now()
```

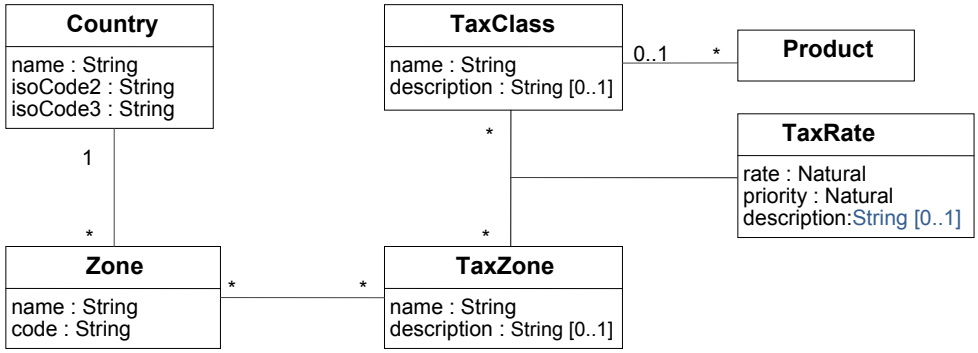
Example test program

```
testprogram CurrencyManagement{
        test CreateCurrency{
                new NewCurrency(title:='Euro', code:='EUR', decimalPlaces:=2) occurs;
        }

        test CreateTheSameCurrencyTwice{
                new NewCurrency(title:='Euro', code:='EUR', decimalPlaces:=2) occurs;
                new NewCurrency(title:='Euro', code:='EUR', decimalPlaces:=2) may not occur;
        }

        test EditCurrency{
                new NewCurrency(title:='Euro', code:='EUR', decimalPlaces:=0) occurs;
                createdCurrency:=Currency.allInstances->select(title='Euro')->any(true);
                new EditCurrency(currency:=createdCurrency,newTitle:='Euro',
                                 newCode:='EUR', newDecimalPlaces:=2) occurs;
                assert equals createdCurrency.decimalPlaces 2;
                //Edition cannot cause duplicates
                euro:=new Currency
                       (title:='Dollar', code:='USD', decimalPlaces:=2, status:=#enabled);
                new EditCurrency(currency:=createdCurrency,newTitle:='Euro',
                                 newCode:='USD', newDecimalPlaces:=2) may not occur;
        }

        test DeleteCurrency{
                euro:=new Currency(title:='Euro', code:='EUR', decimalPlaces:=2);
                //We cannot delete a currency if there is no other currency enabled
                new DeleteCurrency(currency:=euro) may not occur;
                new Currency(title:='Dollar', code:='USD', status:=#enabled);
                new DeleteCurrency(currency:=euro) occurs;
        }

        test ChangeCurrencyStatus{
                euro:=new Currency(title:='Euro', code:='EUR',
                                   decimalPlaces:=2,status:=#disabled);
                new CurrencyStatusChange(currency:=euro, newStatus:=#enabled) occurs;
                assert equals euro.status #enabled;

                //We cannot disable a currency if there is no other currency enabled
                new CurrencyStatusChange(currency:=euro, newStatus:=#disabled) may not occur;
        }

        test SetDefaultCurrency{
                //Initialize store
                franc:=new Currency(title:='Franc', code:='FR');
                french:=new Language(name:='French', code:='FR');
                france:=new Country(name:='France', isoCode2:='FR', isoCode3:='FRA');
                cos:=new OrderStatus;
                cosl:=new OrderStatusInLanguage(language:=french,orderStatus:=cos);
                cosl.name:='annulé';
                dos:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=french);
                dosl.name:='en attenté';
                s:=new Store(name:='CréaPlaisir');
                s.defaultCurrency:=franc;
                s.country:=france;
                s.cancelledStatus:=cos;
                s.defaultStatus:=dos;
                s.defaultLanguage:=french;

                //We test that a new currency is set as default currency
                euro := new Currency(title:='Euro', code:='EUR', decimalPlaces:=2);
                new SetDefaultCurrency(currency:=euro) occurs;
                assert equals s.defaultCurrency euro;
                assert not equals s.defaultCurrency franc;
        }
}
```

9.8. Location & Taxes

Structural schema

In order to supply a flexible use of taxes, product prices are stored tax free. This allows calculating the final price of products depending on the customer's location and the tax class applied to it.



[IC1] A *Country* is identified either by its name or its ISO codes.

context Country::nameAndCodesAreUnique: Boolean
body :
Country.allInstances() -> isUnique (name) **and**
Country.allInstances() -> isUnique (isoCode2) **and**
Country.allInstances() -> isUnique (isoCode3)

[IC2] A *Zone* is identified either by its name and country or its code and country.

context Zone::nameAndCountryAndCodeAndCountryAreUnique: Boolean
body :
Zone.allInstances() -> isUnique (Tuple{n:name, c:country}) **and**
Zone.allInstances() -> isUnique (Tuple{n:code, c:country})

[IC3] A *TaxZone* is identified by its name.

context TaxZone::nameIsUnique: Boolean
body : TaxZone.allInstances() -> isUnique (name)

[IC4] A *TaxClass* is identified by its name

context TaxClass::nameIsUnique: Boolean
body : TaxClass.allInstances() -> isUnique (name)

Use Cases

Add a country

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a country.

Main Success Scenario:

1. The store administrator provides the details of the new country:
   [→*NewCountry*]
2. The system validates that the data is correct.
3. The system saves the new country.

Edit a country

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a country.

Main Success Scenario:

1. The store administrator selects the country to be edited.
2. The store administrator provides the new details of the selected country:
   [→*EditCountry*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a country

Primary Actor: Store administrator

Precondition: The country is not the location of any address.

Trigger: The store administrator wants to delete a country.

Main Success Scenario:

1. The store administrator selects the country to be deleted.

2. The system warns the store administrator of the number of zones which are part of the country to be deleted.
3. The store administrator confirms that he wants to delete the country and their zones:

   [→*DeleteCountry*]
4. The system deletes the country and their zones.

Add a zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a zone.

Main Success Scenario:

1. The store administrator provides the details of the new zone:

   [→*NewZone*]
2. The system validates that the data is correct.
3. The system saves the new zone.

Edit a zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a zone.

Main Success Scenario:

1. The store administrator selects the zone to be edited.
2. The store administrator provides the new details of the selected zone:

   [→*EditZone*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a zone

Primary Actor: Store administrator

Precondition: The zone is not the location of any address.

Trigger: The store administrator wants to delete a zone.

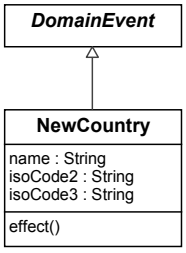
Main Success Scenario:

1. The store administrator selects the zone to be deleted.
2. The store administrator confirms that he wants to delete the zone:

   [→*DeleteZone*]
3. The system deletes the zone.

Events

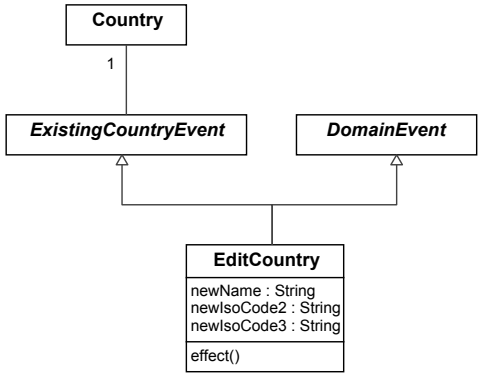
NewCountry



«IniIC»
context NewCountry::countryDoesNotExist(): Boolean
body :
not Country.allInstances() -> exists(c | c.name=self.name **and**
c.isoCode2=self.isoCode2 **and**
c.isoCode3=self.isoCode3)

context NewCountry::effect()
post :
c.oclIsNew() **and**
c.oclIsTypeOf(Country) **and**
c.name = self.name **and**
c.isoCode2 = self.isoCode2 **and** c.isoCode3 = self.isoCode3

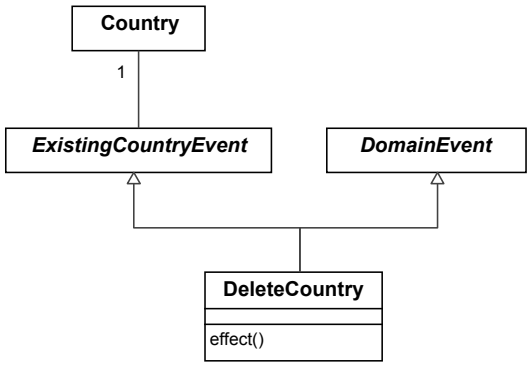
EditCountry



«InilC»
context EditCountry::countryDoesNotExist(): Boolean
 body : (Country.allInstances() - Set{self.country}).name->excludes(self.newName) and
 (Country.allInstances() - Set{self.country}).isoCode2->excludes(self.newIsoCode2)and
 (Country.allInstances() - Set{self.country}).isoCode3->excludes(self.newIsoCode3)

context EditCountry::effect()
 post :
 country.name = self.newName and
 country.isoCode2 = self.newIsoCode2 and
 country.isoCode3 = self.newIsoCode3

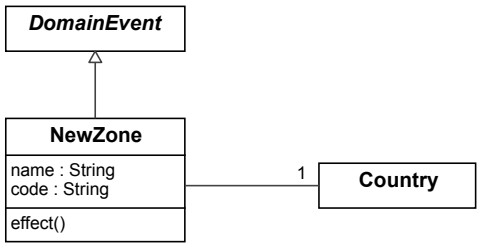
DeleteCountry



«InilC»
context DeleteCountry::countryIsNotALocation():Boolean
 body :
 Store.allInstances() -> any(true).country <> self.country and
 Address.allInstances().country -> excludes(self.country)

context DeleteCountry::effect()
 post : not self.country@pre.oclIsKindOf(OclAny)

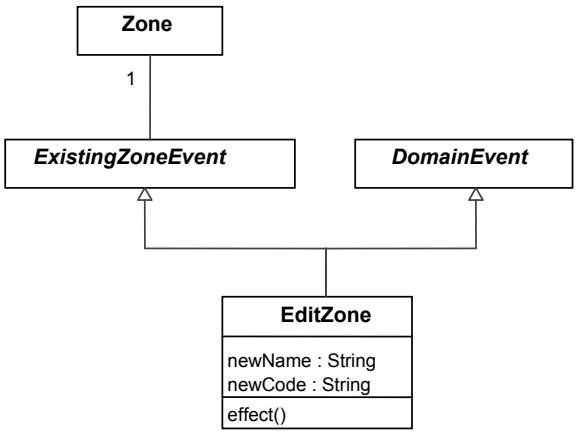
NewZone



«InilC»
context NewZone::ZoneDoesNotExist(): Boolean
 body :
 not Zone.allInstances() -> exists (z | z.name = self.name and z.country = self.country or
 z.code = self.code and z.country = self.country)

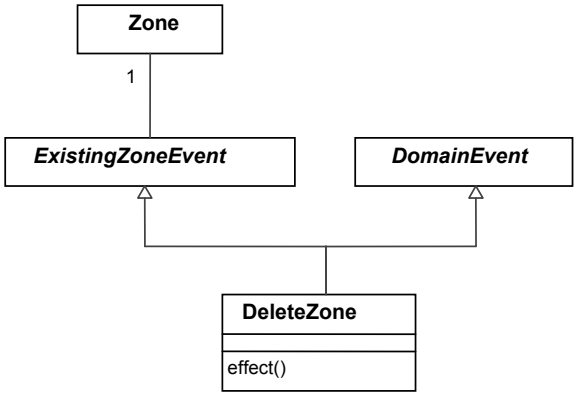
```
context  NewZone::effect()
  post :
     z.oclIsNew()  and
     z.oclIsTypeOf(Zone) and
     z.name = self.name and
     z.code = self.code and
     z.country = self.country
```

EditZone




```
«IniIC»
context EditZone::zoneDoesNotExist(): Boolean
  body :  (Zone.allInstances() - Set{self.zone}).name->excludes(self.newName) and
             (Zone.allInstances() - Set{self.zone}).code->excludes(self.newCode)

context  EditZone::effect()
  post :  self.zone.name = self.newName and  self.zone.code = self.newCode
```

DeleteZone



«IniIC»
context DeleteZone::ZoneIsNotALocation():Boolean
body : Store.allInstances() -> any(true).zone <> self.zone **and**
Address.allInstances().zone -> excludes(self.zone)

context DeleteZone::effect()
post : **not** self.zone@pre.oclIsKindOf(OclAny)
post : self.country@pre.zone -> forAll(z | Zone.allInstances()->excludes(z))

Example test programs

```
testprogram LocationsManagement{

        fixturecomponent DeutschlandCountryCreated{
                de:=new Country(name:='Deutschland', isoCode2:='GE', isoCode3:='DEU');
        }

        test CreateCountry{
                new NewCountry(name:='Deutschland', isoCode2:='DE', isoCode3:='DEU') occurs;
        }

        test CreateTheSameCountryTwice{
                new NewCountry(name:='Deutschland', isoCode2:='DE', isoCode3:='DEU') occurs;
                new NewCountry(name:='Deutschland', isoCode2:='DE',
                                isoCode3:='DEU') may not occur;
        }

        test EditCountry{
                 load DeutschlandCountryCreated;
                 new EditCountry(country:=de,newName:='Deutschland',
                                 newIsoCode2:='DE', newIsoCode3:='DEU') occurs;
                 assert equals de.isoCode2 'DE';
        }

        test DeleteCountryWithoutZones{
                load DeutschlandCountryCreated;
                new DeleteCountry(country:=de) occurs;
        }

        test DeleteTheCountryWhereTheStoreIsLocated{

                //Initialize store
                load DeutschlandCountryCreated;
                mark:=new Currency(title:='Mark', code:='MK');
                deutsch:=new Language(name:='Deutsch', code:='DE');
                cos:=new OrderStatus;
                cosl:=new OrderStatusInLanguage(language:=deutsch,orderStatus:=cos);
                cosl.name:='abgebrochen';
                dos:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=deutsch);
                dosl.name:='unentschieden';
                s:=new Store(name:='Geschenkwelt24');
                s.defaultCurrency:=mark;
                s.country:=de;
                s.cancelledStatus:=cos;
                s.defaultStatus:=dos;
                s.defaultLanguage:=deutsch;

                new DeleteCountry(country:=de) may not occur;
        }

        test CreateZone{
                load DeutschlandCountryCreated;
                new NewZone(country:=de,name:='Waden-Wurttemberg', code:='WW') occurs;
        }
```

```
test CreateTheSameZoneTwice{
        load DeutschlandCountryCreated;
        ww:=new Zone(country:=de,name:='Waden', code:='WW');
        new NewZone(country:=de,name:='Waden-Wurttemberg', code:='WW') may not occur;
}

test EditZone{
        load DeutschlandCountryCreated;
        ww:=new Zone(country:=de,name:='Waden', code:='WW');
        new EditZone(zone:=ww, newName:='Waden-Wurttemberg', newCode:='WW') occurs;
        assert equals ww.name 'Waden-Wurttemberg';
}

test DeleteZone{
        load DeutschlandCountryCreated;
        new NewZone(country:=de,name:='Waden-Wurttemberg', code:='WW') occurs;
        ww:=Zone.allInstances->any(code='WW');
        new DeleteZone(zone:=ww) occurs;
}

test DeleteCountryWithZones{
        load DeutschlandCountryCreated;
        new NewZone(country:=de,name:='Waden-Wurttemberg', code:='WW') occurs;
        new DeleteCountry(country:=de) occurs;
}
```

Use Cases

Add a tax zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a tax zone.

Main Success Scenario:

1. The store administrator provides the details of the new tax zone:

   [→*NewTaxZone*]

2. The system validates that the data is correct.
3. The system saves the new tax zone.

Edit a tax zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a tax zone.

Main Success Scenario:

1. The store administrator selects the tax zone to be edited.
2. The store administrator provides the new details of the selected tax zone:

   [→*EditTaxZone*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a tax zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a tax zone.

Main Success Scenario:

1. The store administrator selects the tax zone to be deleted.
2. The store administrator confirms that he wants to delete the tax zone:

   [→*DeleteTaxZone*]
3. The system deletes the tax zone and all the associated tax rates.

Add a tax class

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a tax class.

Main Success Scenario:

1. The store administrator provides the details of the new tax class:

   [→*NewTaxClass*]
2. The system validates that the data is correct.
3. The system saves the new tax class.

Edit a tax class

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a tax class.

Main Success Scenario:

1. The store administrator selects the tax class to be edited.
2. The store administrator provides the new details of the selected tax class:

   [→*EditTaxClass*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a tax class

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a tax class.

Main Success Scenario:

1. The store administrator selects the tax class to be deleted.
2. The system informs the store administrator about how many products are associated to the deleted tax class.
3. The store administrator confirms that he wants to delete the tax class:

   [→*DeleteTaxClass*]
4. The system deletes the tax class and all the associated tax rates.

Extensions:

2a. The store administrator don't want to delete the tax class.

   2a1. The use case ends.

Add a tax rate

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a tax rate.

Main Success Scenario:

1. The store administrator provides the details of the new tax rate:

   [→*NewTaxRate*]
2. The system validates that the data is correct.
3. The system saves the new tax rate.

Edit a tax rate

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a tax rate.

Main Success Scenario:

1. The store administrator selects the tax rate to be edited.
2. The store administrator provides the new details of the selected tax rate:

   [→*EditTaxRate*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a tax rate

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a tax rate.

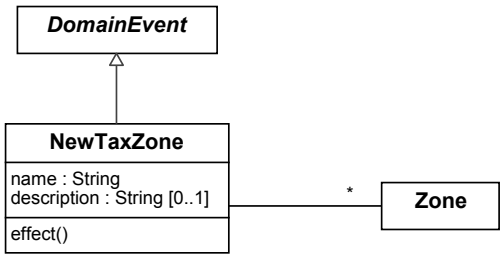
Main Success Scenario:

1. The store administrator selects the tax rate to be deleted.
2. The store administrator confirms that he wants to delete the tax rate:

   [→*DeleteTaxRate*]
3. The system deletes the tax rate.

Events

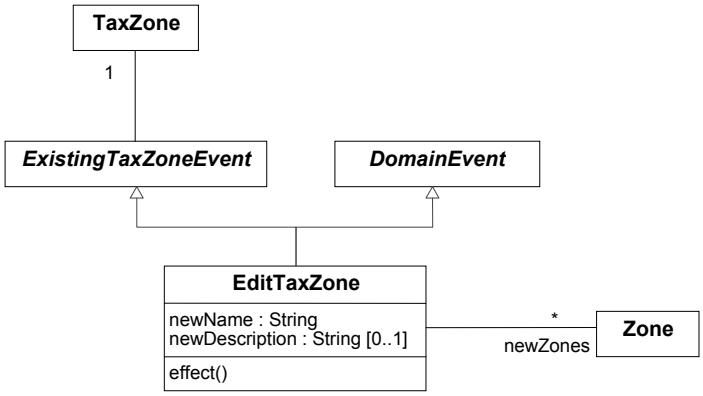
NewTaxZone



«InilC»
context NewTaxZone::TaxZoneDoesNotExist(): Boolean
body : **not** TaxZone.allInstances() -> exists (tz | tz.name = self.name)

context NewTaxZone::effect()
post :
tz.oclIsNew() **and**
tz.oclIsTypeOf(TaxZone) **and**
tz.name = self.name **and**
tz.description = self.description **and**
tz.zone = self.zone

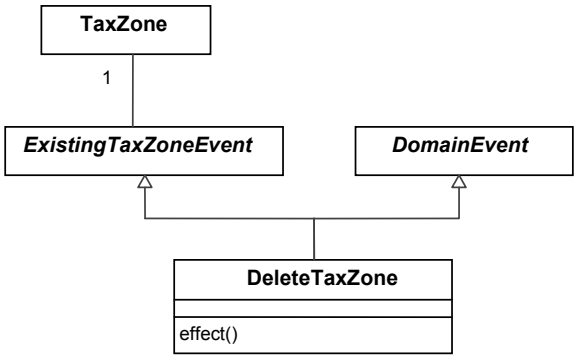
EditTaxZone



«InilC»
context EditTaxZone::TaxZoneDoesNotExist(): Boolean
body : (TaxZone.allInstances() - Set{self.taxZone}).name->excludes(self.newName)

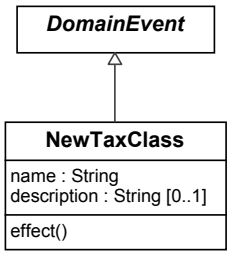
context EditTaxZone::effect()
post :
self.taxZone.name = self.newName **and**
self.taxZone.description = self.newDescription **and**
self.taxZone.zone = self.newZones

DeleteTaxZone



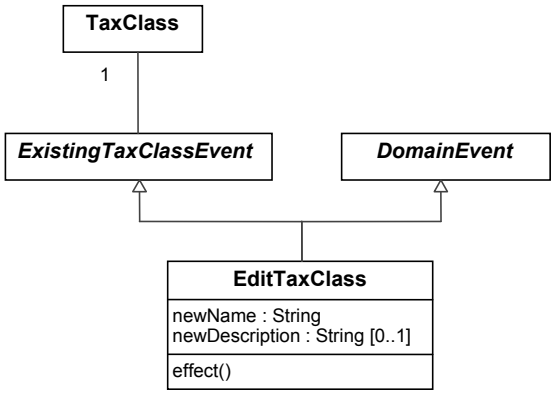

```
context  DeleteTaxZone::effect()
  post deleteTaxZone:
     not self.taxZone@pre.oclIsKindOf(OclAny)
  post deleteAssociatedTaxRates:
     self.taxZone@pre.taxRate@pre -> forAll(tr | tr.oclIsKindOf(OclAny))
```

NewTaxClass



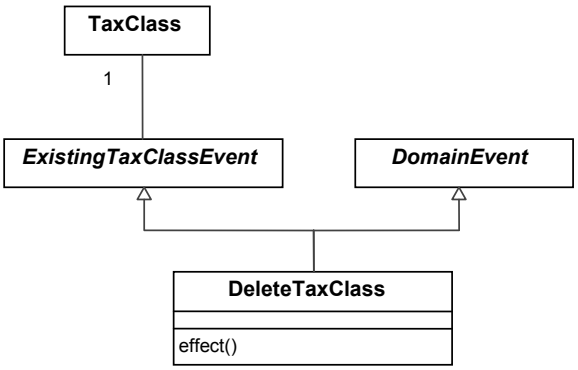

```
«IniIC»
context NewTaxClass::TaxClassDoesNotExist(): Boolean
  body :  not TaxClass.allInstances() -> exists (tc | tc.name = self.name)

context  NewTaxClass::effect()
  post :
     tc.oclIsNew()  and
     tc.oclIsTypeOf(TaxClass) and
     tc.name = self.name and
    tc.description = self.description
```

EditTaxClass



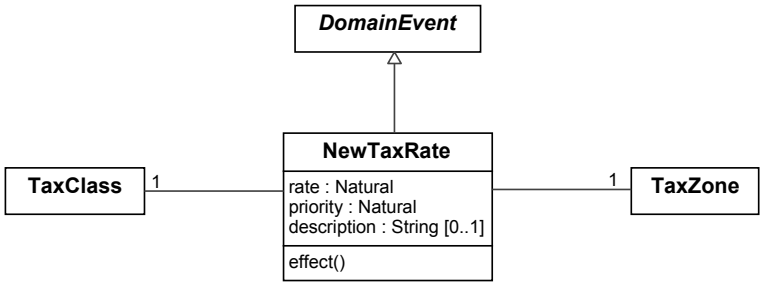

```
«IniIC»
context EditTaxClass::TaxClassDoesNotExist(): Boolean
  body :  (TaxClass.allInstances() - Set{self.taxClass}).name->excludes(self.newName)

context  EditTaxClass::effect()
   post :
      self.taxClass.name = self.newName and  self.taxClass.description = self.newDescription
```

DeleteTaxClass




```
context  DeleteTaxClass::effect()
   post deleteTaxClass:
      not self.taxClass@pre.oclIsKindOf(OclAny)
    post deleteAssociatedTaxRates:
      self.taxClass@pre.taxRate@pre -> forAll(tr | tr.oclIsKindOf(OclAny))
```

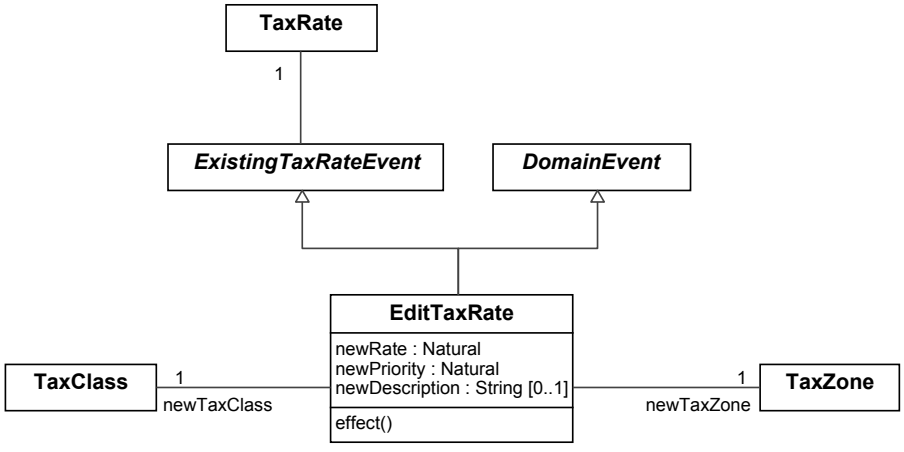
NewTaxRate



«IniIC»
context NewTaxRate::TaxRateDoesNotExist(): Boolean
body :
not TaxRate.allInstances() -> exists (tr | tr.taxClass = self.taxClass **and** tr.taxZone = self.taxZone)

context NewTaxRate::effect()
post :
tr.oclIsNew() **and** tr.oclIsTypeOf(TaxRate) **and**
tr.rate = self.rate **and**
tr.priority = self.priority **and**
tr.description = self.description **and**
tr.taxClass = self.taxClass **and**
tr.taxZone = self.taxZone

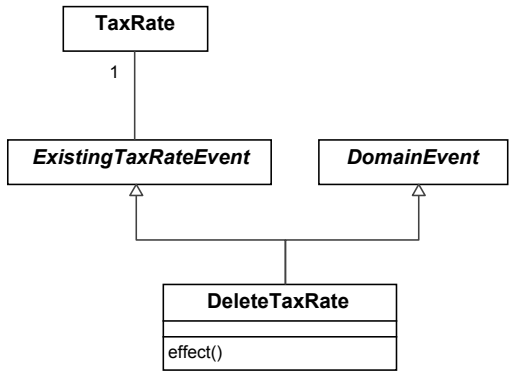
EditTaxRate



«IniIC»
context EditTaxRate::TaxRateDoesNotExist(): Boolean
body : (TaxRate.allInstances - Set{self.taxRate})->select(tr |
tr.taxClass = self.newTaxClass **and** tr.taxZone = self.newTaxZone) -> size()=0

```
context EditTaxRate::effect()
  post :
     self.taxRate.rate = self.newRate and
     self.taxRate.priority = self.newPriority and
     self.taxRate.description = self.newDescription and
     self.taxRate.taxClass = self.newTaxClass and
     self.taxRate.taxZone = self.newTaxZone
```

DeleteTaxRate




```
context DeleteTaxRate::effect()
  post : not self.taxRate@pre.oclIsKindOf(OclAny)
```

Example test programs

```
testprogram TaxesConfigurationManagement{

        spain:=new Country(name:='Spain', isoCode2:='ESP', isoCode3:='ES');
        catalonia:=new Zone(name:='Catalonia', code:='CAT', country:=spain);
        andalucia:=new Zone(name:='Andalucia', code:='AND', country:=spain);
        zones:=spain.zone;

        test AddTaxZone{
                new NewTaxZone(name:='SpanishVAT', zone:=catalonia,andalucia) occurs;
        }

        test EditTaxZone{
                zones:=spain.zone;
                tz:=new TaxZone(name:='SpanishVAT', zone:=zones);
                new EditTaxZone(taxZone:=tz, newName:='SpanishVAT', newZones:=catalonia)
                occurs;
                assert true tz.zone->excludes(andalucia);
                assert true tz.zone->includes(catalonia);
        }

        test DeleteTaxZoneWithoutTaxRates{
                tz:=new TaxZone(name:='SpanishVAT', zone:=zones);
                new DeleteTaxZone(taxZone:=tz) occurs;
        }
```

```
        test DeleteTaxZoneWithTaxRates{
                tz:=new TaxZone(name:='SpanishVAT', zone:=zones);
                tc:=new TaxClass(name:='GeneralVAT');
                tc2:=new TaxClass(name:='ReducedVAT');
                new TaxRate(taxClass:=tc,taxZone:=tz);
                new TaxRate(taxClass:=tc2,taxZone:=tz);
                new DeleteTaxZone(taxZone:=tz) occurs;
        }

        test AddTaxClass{
                new NewTaxClass(name:='SpanishVAT');
                new NewTaxClass(name:='SpanishVAT') may not occur;
        }

        test EditTaxClass{
                tc:=new TaxClass(name:='VAT');
                new EditTaxClass(taxClass:=tc,newName:='GeneralVAT') occurs;
        }

        test DeleteTaxClassWithoutZoneRates{
                tc:=new TaxClass(name:='GeneralVAT');
                new DeleteTaxClass(taxClass:=tc) occurs;
        }

        test DeleteTaxClassWithZoneRates{
                tz:=new TaxZone(name:='SpanishVAT', zone:=zones);
                tc:=new TaxClass(name:='GeneralVAT');
                new TaxRate(taxClass:=tc,taxZone:=tz);
                new DeleteTaxClass(taxClass:=tc) occurs;
        }

        test AddTaxRate{
                tz:=new TaxZone(name:='SpanishVAT', zone:=zones);
                tc:=new TaxClass(name:='GeneralVAT');
                new NewTaxRate(taxClass:=tc, taxZone:=tz, rate:=16, priority:=1) occurs;
        }

        test EditTaxRate{
                tz:=new TaxZone(name:='SpanishVAT', zone:=zones);
                tc:=new TaxClass(name:='GeneralVAT');
                tc2:=new TaxClass(name:='ReducedVAT');
                tr:=new TaxRate(taxClass:=tc,taxZone:=tz);
                tr.rate:=7;
                new EditTaxRate(taxRate:=tr,newTaxClass:=tc2,newTaxZone:=tz,newRate:=7) occurs;
        }

        test DeleteTaxRate{
                tz:=new TaxZone(name:='SpanishVAT', zone:=zones);
                tc:=new TaxClass(name:='GeneralVAT');
                tr:=new TaxRate(taxClass:=tc,taxZone:=tz);
                new DeleteTaxRate(taxRate:=tr) occurs;
        }
}
```

```
testprogram DefaultProductTaxesCalculation{

        /*This test program checks that the default gross
        price  (shown in the online store) of a product is well-calculated. The default
        gross price is calculated by taking into account the
        zone where the store is located*/

        //FIXTURE
        //Languages
        english:=new Language(name:='English', code:='EN');
        spanish:=new Language(name:='Spanish', code:='ES');

        //Currencies
        cad:=new Currency(title:='Canadian Dollar', code:='CAD');
        eur:=new Currency(title:='Euro', code:='EUR');
```

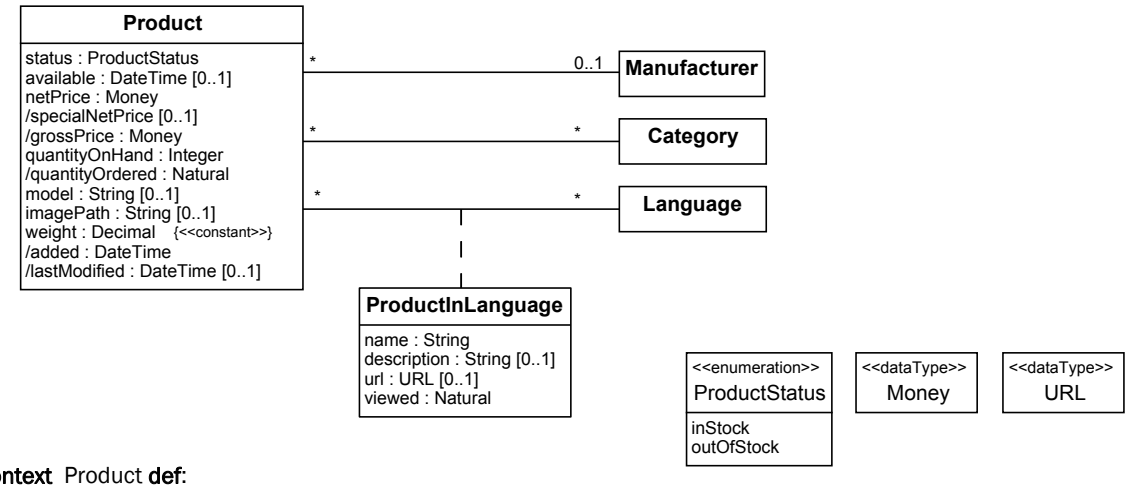
```
//Countries
canada:=new Country(name:='Canada', isoCode2:='CA', isoCode3:='CAN');
spain:=new Country(name:='Spain', isoCode2:='ES', isoCode3:='ESP');

//Zones
andalucia:=new Zone(name:='Andalucia', code:='AND', country:=spain);
ontario:=new Zone(name:='Ontario', code:='ONT', country:=canada);
quebec:=new Zone(name:='Quebec', code:='QUE', country:=canada);

//Order Status
cos:=new OrderStatus;
cosInEnglish:=new OrderStatusInLanguage(language:=english,orderStatus:=cos);
cosInEnglish.name:='Cancelled';
cosInSpanish:=new OrderStatusInLanguage(language:=spanish,orderStatus:=cos);
cosInSpanish.name:='Cancelado';
dos:=new OrderStatus;
dosInEnglish:=new OrderStatusInLanguage(orderStatus:=dos, language:=english);
dosInEnglish.name:='Pending';
dosInSpanish:=new OrderStatusInLanguage(orderStatus:=dos, language:=spanish);
dosInEnglish.name:='Pendiente';

//FIXTURE COMPONENTS
/*We create two different shop configurations:
A canadian store (with only one tax class)
An spanish store (with three different tax classes)
We apply them in the test cases two check the gross
price calculation in different tax configurations*/

fixturecomponent CanadianStoreInitialization{
        //Store initialization
        s:=new Store(name:='CanadianStore');
        s.defaultLanguage:=english;
        s.defaultCurrency:=cad;
        s.country:=canada;
        s.cancelledStatus:=cos;
        s.defaultStatus:=dos;

        //Tax configuration
        //We create a tax zone for Canada
        TaxZone canadaFederalTaxes:=new TaxZone(name:='Canada Federal Taxes');
        canadaFederalTaxes.zone:=quebec,ontario;

        //We create an specific tax zone for Quebec
        TaxZone quebecLocalTaxes:=new TaxZone(name:='QuebecLocalTaxes');
        quebecLocalTaxes.zone:=quebec;

        //We consider a single tax class
        TaxClass general:=new TaxClass(name:='general');

        //For each TaxClass, there is a different tax rate applied in each zone
        TaxRate canadianFederalTaxRate:=new TaxRate
                        (taxClass:=general, taxZone:=canadaFederalTaxes);
        canadianFederalTaxRate.rate:=7;
        canadianFederalTaxRate.priority:=1;

        quebecLocalTaxRate:=new TaxRate(taxClass:=general, taxZone:=quebecLocalTaxes);
        quebecLocalTaxRate.rate:=7.5;
        quebecLocalTaxRate.priority:=2;
}

fixturecomponent SpanishStoreInitialization{
        //Store initialization
        s:=new Store(name:='SpanishStore');
        s.defaultLanguage:=spanish;
        s.defaultCurrency:=cad;
        s.country:=spain;
        s.cancelledStatus:=cos;
        s.defaultStatus:=dos;

        //We create a specific tax zone
        TaxZone spanishVAT:=new TaxZone(name:='SpanishVAT',
```

```
                description:='This zone includes all VAT varieties applied in Spain');
            spanishVAT.zone:=andalucia;

            //In Spain there are three types of VAT: general VAT (16%),
           //reduced VAT(7%) and super-reduced VAT(4%)
            TaxClass general:=new TaxClass(name:='General VAT');
            TaxClass reduced:=new TaxClass(name:='ReducedVAT');
            TaxClass superreduced:=new TaxClass(name:='Super-reduced VAT');

            //For each TaxClass, there is a different tax rate applied in each zone
            TaxRate generalRate:=new TaxRate(taxClass:=general, taxZone:=spanishVAT);
            generalRate.rate:=16;
            generalRate.priority:=1;

            TaxRate reducedRate:=new TaxRate(taxClass:=reduced, taxZone:=spanishVAT);
            reducedRate.rate:=7;
            reducedRate.priority:=1;

            TaxRate superReducedRate:=new TaxRate
                                (taxClass:=superreduced, taxZone:=spanishVAT);
            superReducedRate.rate:=4;
            superReducedRate.priority:=1;
    }

    test DefaultGrossPriceWithDifferentTaxClasses{
            load SpanishStoreInitialization;

            //We locate the store in the zone Andalucia
            s.zone := andalucia;

            //The reduced VAT is applied to cultural events, among others products
            Product greaseMusicalAdmission:=new Product(netPrice:=50);
            greaseMusicalAdmission.taxClass:=reduced;
            assert equals greaseMusicalAdmission.grossPrice() 53.5;

            //The super-reduced VAT is applied to books, among other products
            Product angelsAndDemonsBook:= new Product(netPrice:=25);
            angelsAndDemonsBook.taxClass:=superreduced;
            assert equals angelsAndDemonsBook.grossPrice() 26.0;

            //The general VAT is applied to those products which are not basic needs or
           //cultural products
            Product whiteWineBottle:= new Product(netPrice:=11);
            whiteWineBottle.taxClass:=general;
            assert equals whiteWineBottle.grossPrice() 12.76;
    }

    test DefaultGrossPriceInDifferentShopLocations{
            /*We test that the gross price (netPrice + taxes) of
            a product is different depending on the store location and the
            taxes configuration.*/

            load CanadianStoreInitialization;

            //We create the example product
            Product theDaVinciCodeBook:= new Product(netPrice:=50);
            theDaVinciCodeBook.taxClass:=general;

            //First, we locate the store in the zone Ontario
            s.zone:=ontario;
            assert equals theDaVinciCodeBook.grossPrice() 53.5;

            /*If the store is located in Quebec, the gross price
            also takes into account the Quebec Local Tax which is
            compounded with the Federal Tax*/
            s.zone:=quebec;
            assert equals theDaVinciCodeBook.grossPrice() 57.5125;
    }
}
```

9.9. Products

Structural schema

The system must know the information about the products offered by the online store.




```
context  Product def:
   addTaxes(z:Zone, basePrice:Money) : Money =
      let appliedTaxRates:Set(TaxRate)=
      z.taxZone.taxRate -> select (tr | tr.taxClass = self.taxClass)
      in
         let  priorities:Set(Natural) =
            if appliedTaxRate -> isEmpty() then set{}
            else appliedTaxRates -> sortedBy(priority).priority -> asSet()
            endif
         in
            if priorities -> isEmpty() then basePrice
            else priorities -> iterate (p:Natural; res:Money = 0 |
                                        res +
                                        (((appliedTaxRates -> select (tr | tr.priority = p).rate
                                         -> sum()) / 100)+1)*basePrice)
            endif
```

[DR1] *Product::**grossPrice*** is the product's *netPrice* taking into account the applied taxes.

```
context Product::grossPrice(): Money
body :  self.addTaxes(Store.allInstances() -> any(true).zone, self.netPrice)
```

[DR2] *Product::**specialNetPrice*** is the special price, if the product is an active special.

```
context Product::specialNetPrice(): Money
body :
   if self.oclIsTypeOf(Special) then
      if self.oclAsType(Special).specialStatus=Status::enabled and
        self.oclAsType(Special).expiryDate < Now()
      then self.oclAsType(Special).specialPrice
      else set{}
```

```
    endif
  else set{}
  endif
```

[DR3] *Product::**added*** is the *DateTime* of product creation.

```
context Product::added(): DateTime
  body : Now()
```

[IC1] A product is identified by a name in a language.

```
context Language::nameIsUnique(): Boolean
 body :
     Language.allInstances->forAll(l |
          l.productInLanguage->isUnique(name))
```

Use cases

Add a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a product to the store catalog.

Main Success Scenario:

1. The store administrator selects the product category.
2. The store administrator provides the product data:

   [→*NewProduct*]
3. The system validates that the data is correct.
4. The system saves the new product.
5. The store administrator provides a product attribute:

   [→*NewProductAttribute*]
6. The system validates that the product attribute is correct.
7. The system saves the new product attribute.

   The store administrator repeats steps 5-7 until he is done.

Extensions:

5a. The product does not have product attributes:

   5a1. The use case ends.

5b. The product option is new:

5b1. Add a product option.

5c. The product option value is new:

5c1. Add a product option value.

Edit a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a product.

Main Success Scenario:

1. The store administrator selects the product to be edited.
2. The store administrator provides the new values for the attributes of the product:

   [→*EditProduct*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a product.

Main Success Scenario:

1. The store administrator selects the product to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the product:

   [→*DeleteProduct*]
4. The system deletes the product and their product attributes.

Extensions:

3a. The product is part of an order:

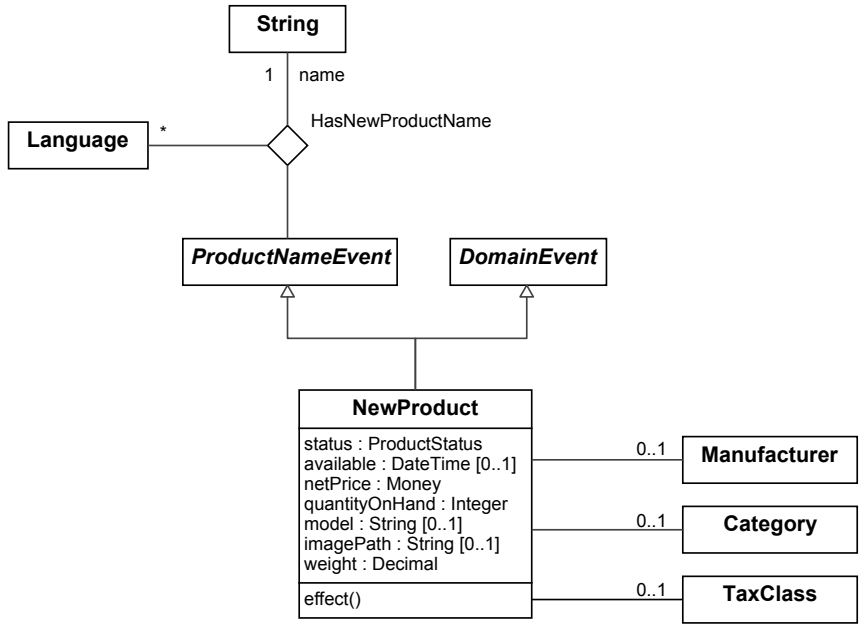
3a1. The system changes the status of the product to out of stock.

[→*ProductStatusChange*]

3a2. The use case ends.

Events

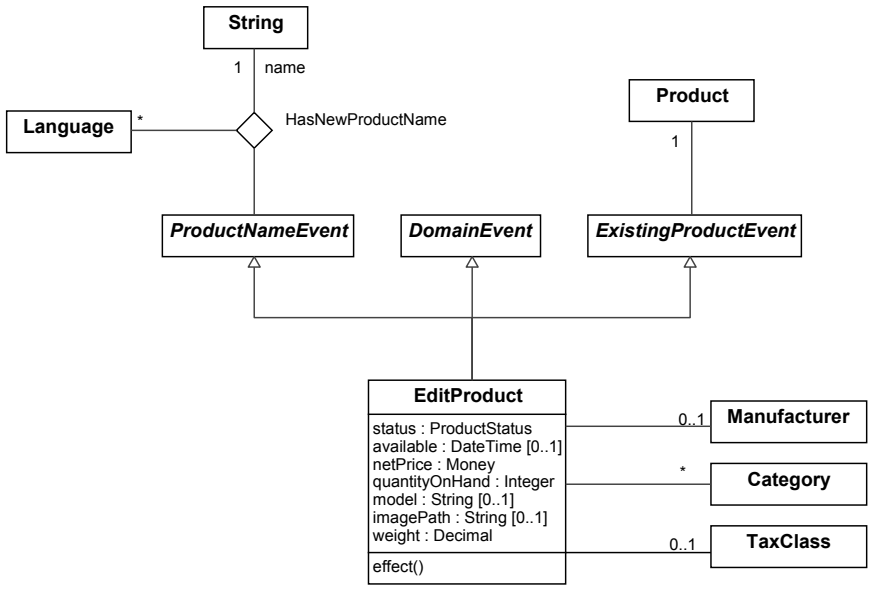
NewProduct



«IniIC»

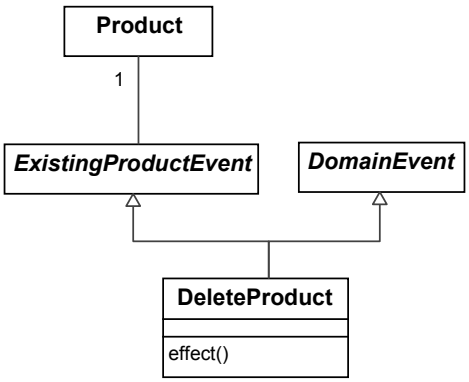
```
context NewProduct::productDoesNotExist(): Boolean
  body :
     Language.allInstances() -> forAll ( l |
           l.productInLanguage.name
           -> excludes( self.hasNewProductName -> select(language=l).name))

context  NewProduct::effect()
  post :
     p.oclIsNew()  and
     p.oclIsTypeOf(Product) and
     p.status = self.status and
     p.available = self.available and
     p.netPrice = self.netPrice and
     p.quantityOnHand = self.quantityOnHand and
     p.model = self.model and
     p.imagePath = self.imagePath and
     p.weight = self.weight and
     p.category = Set{self.category} and
     p.manufacturer = self.manufacturer and
     p.taxClass = self.taxClass and
     Language.allInstances() ->
        forAll (l|
          self.hasNewProductName -> select(language=l).name =
          p.productInLanguage->select(language=l).name)
```

EditProduct



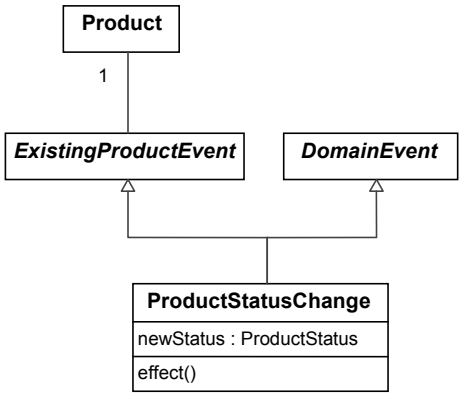

```
«IniIC»
context EditProduct::productDoesNotExist(): Boolean
  body: Language.allInstances() -> forAll ( l |
          l.productInLanguage.name
            -> excludes(self.hasNewProductName -> any(languageOfProduct=l).nameOfProduct) or
          (self.hasNewProductName->any(languageOfProduct=l).nameOfProduct =
          self.product.productInLanguage->any(language=l).name))


context  EditProduct::effect()
  post :
     self.product.status = self.status and
     self.product.available = self.available and
     self.product.netPrice = self.netPrice and
     self.product.quantityOnHand = self.quantityOnHand and
     self.product.model = self.model and
     self.product.imagePath = self.imagePath and
     self.product.weight = self.weight and
     self.product.manufacturer = self.manufacturer and
     self.product.category = self.category and
     self.product.taxClass = self.taxClass and
     Language.allInstances()
        -> forAll (l|
          self.hasNewProductName -> select(language=l).name =
          self.product.productInLanguage->select(language=l).name)
  post :
     self.product.lastModified = Now()
```

DeleteProduct




```
context DeleteProduct::effect()
 post:
        if product@pre.orderLine -> size()=0
           then Product.allInstances->excludes(product@pre)
         else
           psc.oclIsNew() and
           psc.oclIsTypeOf(ProductStatusChange) and
           psc.newStatus = Status::outOfStock and
           psc.product = self.product@pre
        endif
```

ProductStatusChange




```
context ProductStatusChange::effect()
 post : self.product.status = self.newStatus
```

Example test programs

```
testprogram AddNewProducts{

        //Test cases are based on a multilingual online shop with two languages
        italian := new Language(name:='Italian', code:='IT');
        english := new Language(name:='English', code:='EN');

        test NewProductWithoutNames{
                new NewProduct(netPrice:=30,quantityOnHand:=50) may not occur;
        }

        test NewProductWithoutNamesForSomeLanguages{
                //We should specify the product name in each language
                s:=new StringDT(string:='Extra Virgin Oil Jar');
                np:=new NewProduct(netPrice:=10,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=s,
                      languageOfProduct:=english,productNameEvent:=this));
                np may not occur;
        }

        test NewProductWithAllNamesSpecified{
                //We test a valid invocation of the event
                englishName:=new StringDT(string:='Extra Virgin Oil Jar');
                italianName:=new StringDT(string:='Giara di olio');
                np:=new NewProduct(netPrice:=10,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=italianName,
                             languageOfProduct:=italian,productNameEvent:=this);
                new HasNewProductName(nameOfProduct:=englishName,
                             languageOfProduct:=english,productNameEvent:=this))
                np occurs;
                createdProduct := Product.allInstances
                                   ->any(productInLanguage
                                   ->exists(name='Extra Virgin Oil Jar'));

                //Although postconditions are checked,
                //we ensure that we can get the product name in each language
                assert equals createdProduct.productInLanguage->any(language=english).name
                             'Extra Virgin Oil Jar';
                assert equals createdProduct.productInLanguage->any(language=italian).name
                             'Giara di olio';
        }


        test NewProductWithEqualNamesInSomeLanguages{
                //osCommerce allows the same product name for different languages
                s:=new StringDT(string:='Lemoncello');
                np:=new NewProduct(netPrice:=30,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=s,
                    languageOfProduct:=italian,productNameEvent:=this);
                new HasNewProductName(nameOfProduct:=s,
                    languageOfProduct:=english,productNameEvent:=this));
                np occurs;
        }

        test NewProductThatAlreadyExists{
                //IB state with a product
                acetoAromatizzato:=new Product(netPrice:=4, quantityOnHand:=70);
                productInItalian:=new ProductInLanguage
                                   (product:=acetoAromatizzato, language:=italian);
                productInItalian.name:='Aceto aromatizzato';
                productInEnglish:=new ProductInLanguage
                                   (product:=acetoAromatizzato, language:=english);
                productInEnglish.name:='Spicy wine vinegar';

                //The creation of a product with the same name in at least one
                //language should not occur
                italianName:=new StringDT(string:='Aceto aromatizzato');
                englishName:=new StringDT(string:='Spicy wine vinegar');
                differentName:=new StringDT(string:='AnyName');
```

```
                np:=new NewProduct(netPrice:=10,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=italianName,
                        languageOfProduct:=italian,productNameEvent:=this);
                new HasNewProductName(nameOfProduct:=differentName,
                        languageOfProduct:=english,productNameEvent:=this));
                np may not occur;
                np2:=new NewProduct(netPrice:=10,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=differentName,
                        languageOfProduct:=italian,productNameEvent:=this);
                new HasNewProductName(nameOfProduct:=englishName,
                        languageOfProduct:=english,productNameEvent:=this));
                np2 may not occur;
                np3:=new NewProduct(netPrice:=10,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=italianName,
                        languageOfProduct:=italian,productNameEvent:=this);
                new HasNewProductName(nameOfProduct:=englishName,
                        languageOfProduct:=english,productNameEvent:=this))
                np3 may not occur;
        }
}
```

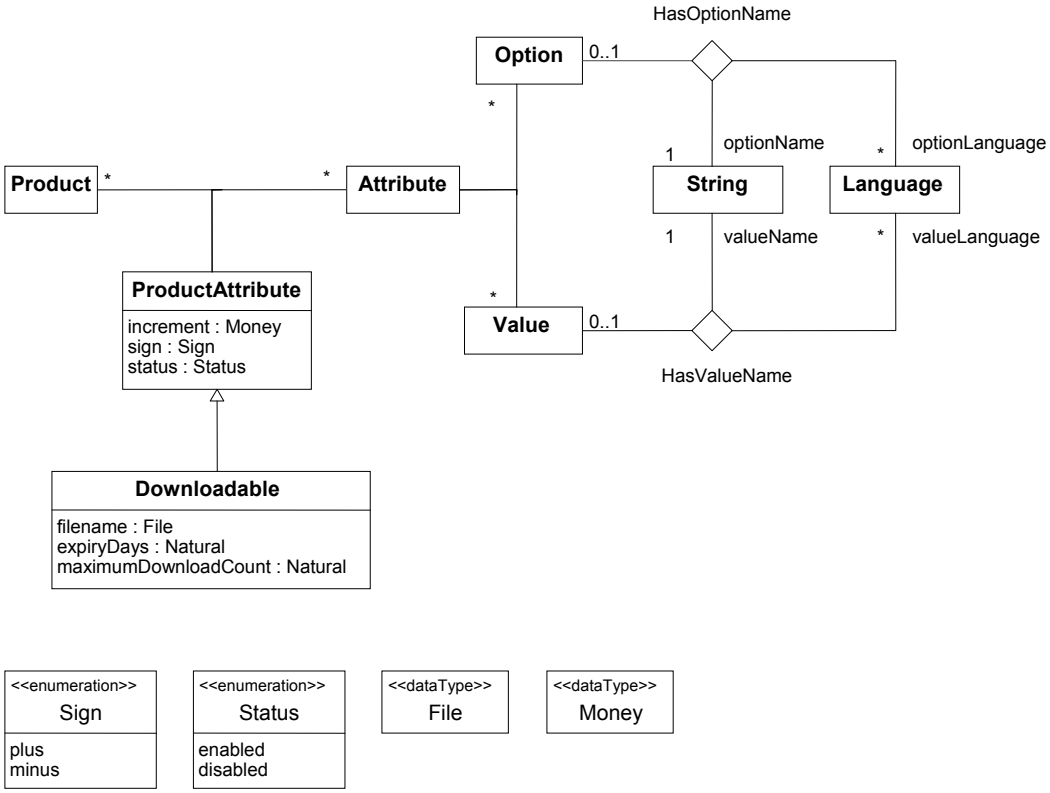
```
testprogram EditProducts{

        english := new Language(name:='English', code:='EN');
        necklace:=new Product(netPrice:=4, quantityOnHand:=70, status:=#outOfStock);
        productInEnglish:=new ProductInLanguage(product:=necklace, language:=english);
        productInEnglish.name:='Necklace';

        test EditProductStatus{
                englishName:=new StringDT(string:='Necklace');
                ep:=new EditProduct(product:=necklace,status:=#inStock,
                                netPrice:=10,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=englishName,
                        languageOfProduct:=english,productNameEvent:=this));
               ep occurs;
                assert equals necklace.status #inStock;
        }

        test EditProductNameInALanguage{
                englishName:=new StringDT(string:='GoldNecklace');
                ep:=new EditProduct(product:=necklace,status:=#inStock,
                                netPrice:=10,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=englishName,
                         languageOfProduct:=english,productNameEvent:=this));
                ep occurs;
        }

        test UnapplicableProductEdition{
                //IB state with a product
                goldnecklace:=new Product(netPrice:=4, quantityOnHand:=70, status:=#inStock);
                productInEnglish:=new ProductInLanguage
                                 (product:=goldnecklace, language:=english);
                productInEnglish.name:='Gold Necklace';

                //A product edition the effect of which violates the product identification
               //constraint cannot occur
                englishName:=new StringDT(string:='GoldNecklace');
                ep:=new EditProduct(product:=necklace,status:=#inStock,
                                netPrice:=10,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=englishName,
                        languageOfProduct:=english,productNameEvent:=this));
                ep occurs;
        }
}
```

```
testprogram DeleteProduct{

        english := new Language(name:='English', code:='EN');
        necklace:=new Product(netPrice:=4, quantityOnHand:=70, status:=#outOfStock);

        productInEnglish:=new ProductInLanguage(product:=necklace, language:=english);
        productInEnglish.name:='Necklace';

        test DeleteProductNotSoldYet{
                new DeleteProduct(product:=necklace) occurs;
                assert true Product.allInstances->excludes(necklace);
        }

        test DeleteSoldProduct{
                //We create an order
                ol:= new OrderLine(product:=necklace,order:=o);
                dollar:=new Currency(title:='USDollar', code:='USD');
                dos:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=english);
                dosl.name:='pending';
                osc := new OrderStatusChange(order:=o,orderStatus:=dos);
                sm:= new FlatRate(status:=#enabled);
                pm:= new Nochex(status:=#enabled);
                usa:=new Country(name:='United States', isoCode2:='US', isoCode3:='USA');
                a:= new Address(country:=usa);
                c := new Customer(address:=a,primary:=a);
                o:= new Order(customer:=c, currency:=dollar,
                              shippingMethod:=sm, paymentMethod:=pm);
                new DeleteProduct(product:=necklace) occurs;
                assert true Product.allInstances->includes(necklace);
                assert equals necklace.status #outOfStock;
        }
}
```

9.10. Product attributes and options

Structural schema

osCommerce allows defining several attributes for each product. Product attributes are used to offer multiple options of a product.



[IC1] In each language, each product option has a unique name.

context Language::optionNameIsUnique(): Boolean
body : self.hasOptionName -> isUnique(optionName)

[IC2] In each language, each product value has a unique name.

context Language::valueNameIsUnique(): Boolean
body : self.hasOptionValue -> isUnique(valueName)

Use cases

Add a product option

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a product option to the store catalog.

Main Success Scenario:

1. The store administrator provides the product option data:
   [→*NewProductOption*]
2. The system validates that the data is correct.
3. The system saves the new product option.

Edit a product option

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a product option.

Main Success Scenario:

1. The store administrator selects the product option to be edited.
2. The store administrator provides the new details of the selected product option:
   [→*EditProductOption*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a product option

Primary Actor: Store administrator

Precondition: The product option has no associated products.

Trigger: The store administrator wants to delete a product option.

Main Success Scenario:

1. The store administrator selects the product option to be deleted.

2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the product option:

   [→*DeleteProductOption*]
4. The system deletes the product option and its associated values if they are not values of other options.

Add a product option value

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a value to a product option.

Main Success Scenario:

1. The store administrator selects the product option.
2. The store administrator provides the product option value data:

   [→*NewProductOptionValue*]
3. The system validates that the data is correct.
4. The system saves the new product option value.

Edit a product option value

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a product option value.

Main Success Scenario:

1. The store administrator selects the product option value to be edited.
2. The store administrator provides the new details of the selected product option value:

   [→*EditProductOptionValue*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a product option value

Primary Actor: Store administrator

Precondition: The product option value has not products linked to it.

Trigger: The store administrator wants to delete a product option value.

Main Success Scenario:

1. The store administrator selects the product option value to delete.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the product option value:
   [→*DeleteProductOptionValue*]
4. The system deletes the product option value.

Add a product attribute

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to assign an attribute to a product.

Main Success Scenario:

1. The store administrator selects the product.
2. The store administrator provides the attribute and the product attribute data (increment and sign):
   [→*NewProductAttribute*]
   [→*NewDownloadableProductAttribute*]
3. The system validates that the data is correct.
4. The system saves the new product attribute.

Extensions:

2a. The product option is new:
   2a1. Add a product option.
2b. The product option value is new:
   2b1. Add a product option value.

Edit a product attribute

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a product attribute.

Main Success Scenario:

1. The store administrator selects the product attribute to be edited.
2. The store administrator provides the new details for the product attribute:

   [→*AttributeChange*]

   [→*IncrementAndSignAttributeChange*]

   [→*EditDownloadableAttribute*]
3. The system validates that the data is correct.
4. The system saves the changes.

The system repeats steps 2-4 until he is done.

Delete a product attribute

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a product attribute.

Main Success Scenario:

1. The store administrator selects the product attribute to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the product attribute:

   [→*DeleteProductAttribute*]
4. The system deletes the product attribute.

Extensions:

3a. The product attribute is part of an existing order line:

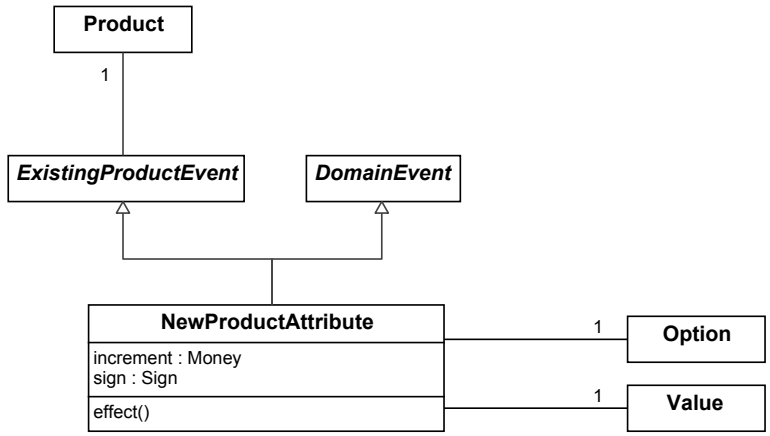
   3a1. The system changes the status of the product attribute to disable.

      [→*ProductAttributeStatusChange*]

   3a2. The use case ends

Events

NewProductAttribute

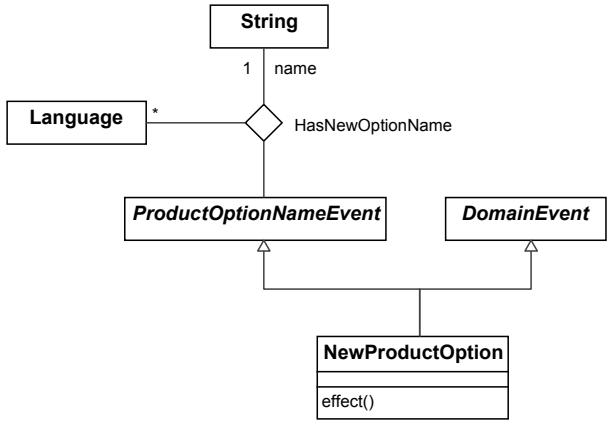


«InilC»
context NewProductAttribute::productAttributeDoesNotExist(): Boolean
body :
not self.product.productAttribute ->
exists(attribute.value=self.value **and**
attribute.option = self.option)

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

context NewProductAttribute::effect()
post :
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

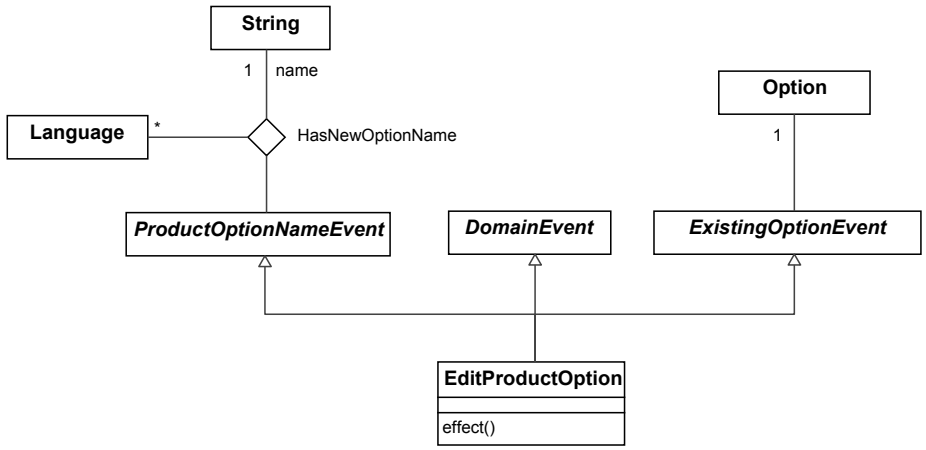
NewProductOption



«IniIC»
context NewProductOption::productOptionDoesNotExist(): Boolean
body :
Language.allInstances() -> forAll (l |
l.hasOptionName.optionName
-> excludes(self.hasNewOptionName -> select(language=l).name))

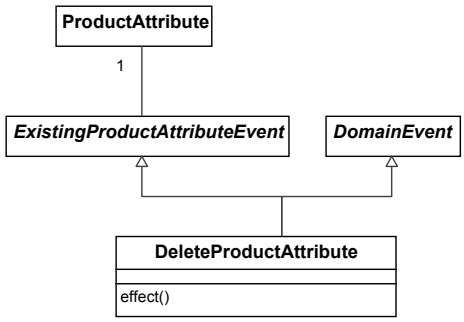
context NewProductOption::effect()
post :
po.oclIsNew() **and**
po.oclIsTypeOf(Option) **and**
Language.allInstances() ->
forAll (l| self.hasNewOptionName -> select(language=l).name =
po.hasOptionName->select(optionLanguage=l).optionName)

EditProductOption



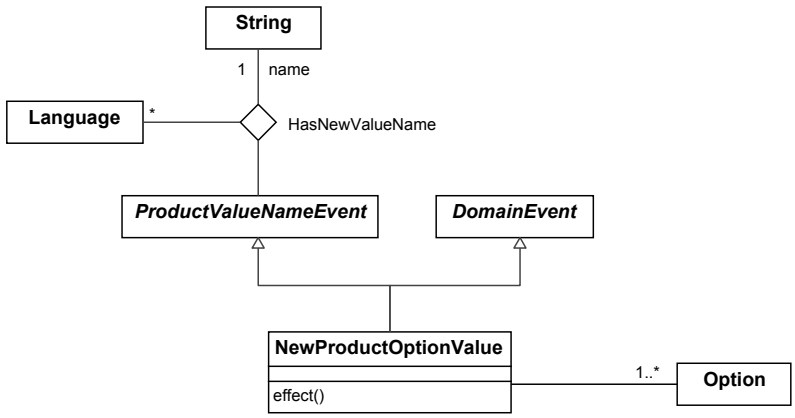
```
«InilC»
context EditProductOptionproduct::OptionDoesNotExist(): Boolean
  body: Language.allInstances -> forAll ( l |
            l.hasOptionName.optionName
              -> excludes(self.hasNewOptionName -> any(languageOfOption=l).nameOfOption) or
             (self.hasNewOptionName->any(languageOfOption=l).nameOfOption =
              self.option.hasOptionName->any(optionLanguage=l).optionName))

context  EditProductOption::effect()
  post :
    Language.allInstances() ->
      forAll (l| self.hasNewOptionName -> select(language=l).name =
                 option.hasOptionName->select(language=l).optionName)
```

DeleteProductAttribute




```
context  DeleteProductAttribute::effect()
  post:  if OrderLineAttribute.allInstances() -> exists(ola |
              ola.attribute=productAttribute.attribute and
              ola.orderLine.product=productAttribute.product)
         then   productAttribute.status=Status::disabled
         else   ProductAttribute.allInstances->excludes(productAttribute@pre)
         endif
```

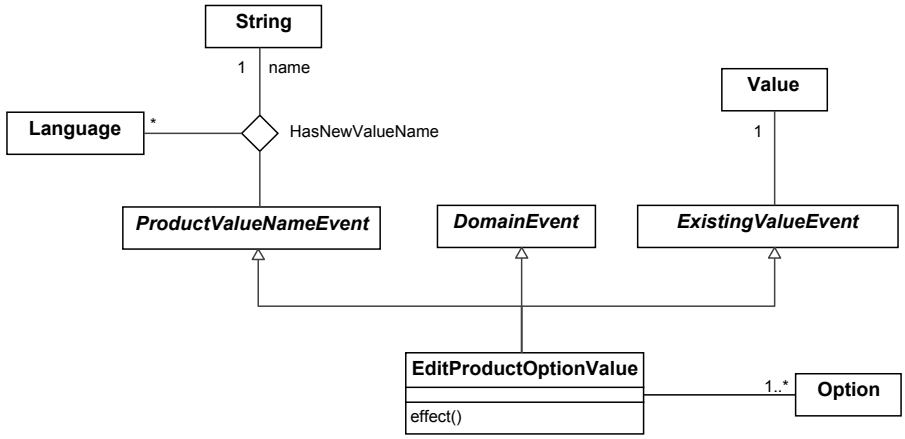
NewProductOptionValue



«IniIC»
context NewProductOptionValue::optionValueDoesNotExist(): Boolean
body :
Language.allInstances() -> forAll (l |
l.hasValueName.valueName
-> excludes(self.hasNewValueName -> select(language=l).name))

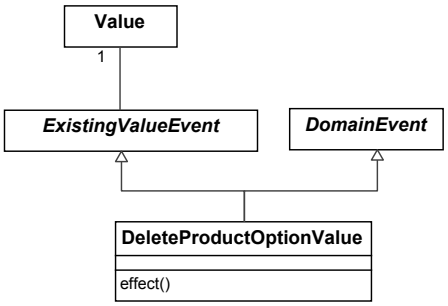
context NewProductOptionValue::effect()
post :
pov.oclIsNew() **and**
pov.oclIsTypeOf(Value) **and**
Language.allInstances() ->
forAll (l| self.hasNewValueName -> select(language=l).name =
pov.hasValueName->select(valueLanguage=l).valueName) **and**
pov.option = self.option

EditProductOptionValue



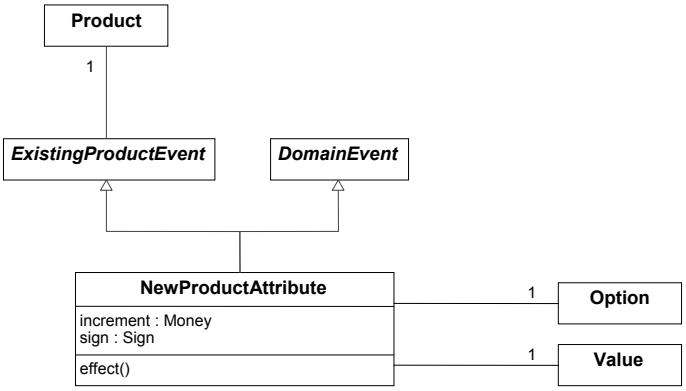
«IniIC»
```
context EditProductOptionValue ::productOptionValueDoesNotExist(): Boolean
  body: Language.allInstances() -> forAll ( l |
                        l.hasValueName.valueName
                         -> excludes(self.hasNewValueName -> any(language=l).name) or
                        (self.hasNewValueName->any(language=l).name =
                        self.value.hasValueName->any(valueLanguage=l).valueName))

context  EditProductOptionValue::effect()
  post :
     Language.allInstances() ->
        forAll (l| self.hasNewValueName -> select(language=l).name =
                     value.hasValueName->select(language=l).valueName) and
     self.value.option = self.option
```

DeleteProductOptionValue



«IniIC»
```
context  DeleteProductOptionValue::HasNotProducts():Boolean
  body : self.value.attribute.product -> isEmpty() and self.value.attribute.orderLineAttribute->isEmpty()

context  DeleteProductOptionValue::effect()
  post :  not self.value@pre.oclIsKindOf(OclAny)
```

NewProductAttribute

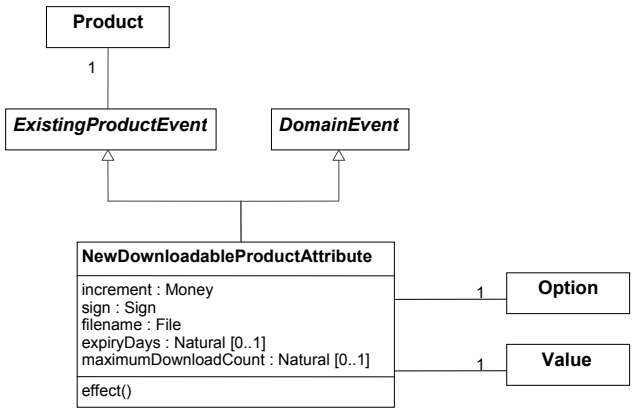


«IniIC»
context NewProductAttribute::productAttributeDoesNotExist(): Boolean
body :
not self.product.productAttribute ->
exists(attribute.value=self.value **and**
attribute.option = self.option)

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

context NewProductAttribute::effect()
post :
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

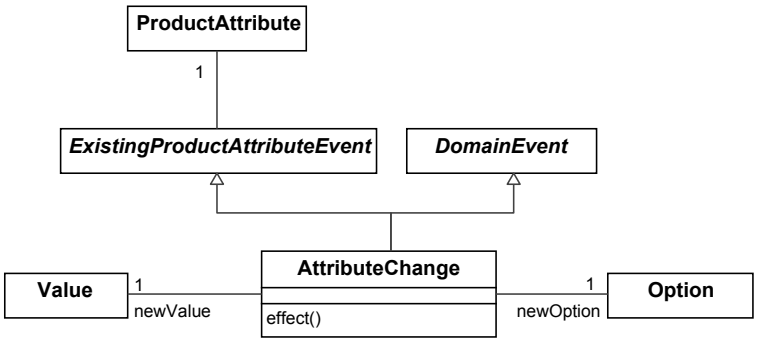
NewDownloadableProductAttribute



«IniIC»

```
context NewDownloadableProductAttribute::productAttributeDoesNotExist(): Boolean
  body :
     not ProductAttribute.allInstances() -> exists (pa | pa.attribute.option = self.option and
                                                          pa.attribute.value = self.value and
                                                          pa.product = self.product)

context  NewDownloadableProductAttribute::effect()
   post :
      dpa.oclIsNew()  and
      dpa.oclIsTypeOf(Downloadable) and
      dpa.increment = self.increment and
      dpa.sign = self.sign and
      dpa.filename = self.filename and
      dpa.product = self.product and
      dpa.attribute.option=self.option and
      dpa.attribute.value=self.value and
      if self.expiryDays.notEmpty() then dpa.expiryDays = self.expiryDays
      else self.expiryDays = Download.daysExpiryDelay
      endif
      and
      if self.maximumDownloadCount .notEmpty() then
         dpa.maximumDownloadCount = self.maximumDownloadCount
      else self.maximumDownloadCount = Download.maximumNumberOfDownloads
      endif
```

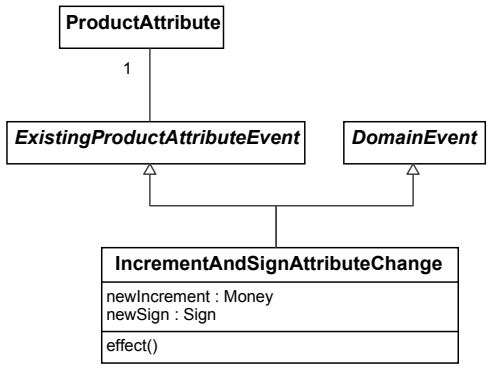
AttributeChange



«IniIC»
context AttributeChange::OptionAndValueAreAValidAttribute(): Boolean
body: Attribute.allInstances()->exists(a| a.option=self.newOption and a.value=self.newValue)

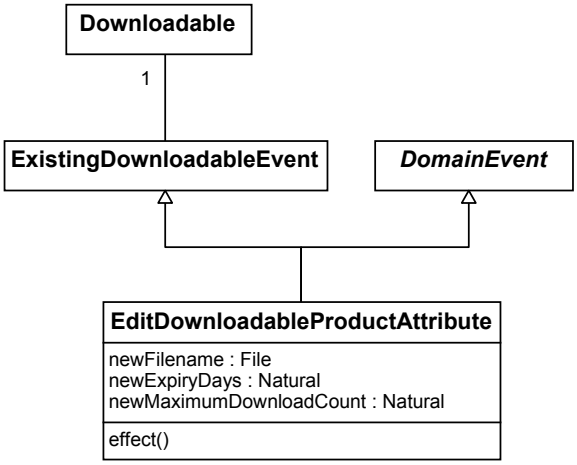
context AttributeChange::effect()
post :
self.productAttribute.attribute.value = self.newValue **and**
self.productAttribute.attribute.option = self.newOption

IncrementAndSignAttributeChange



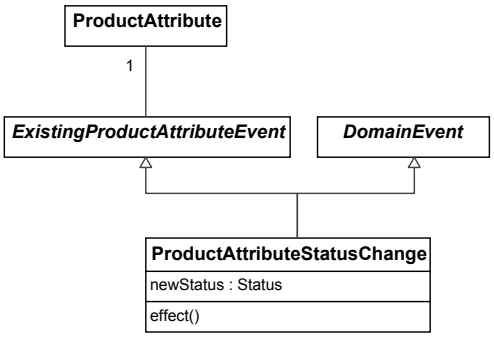
context IncrementAndSignAttributeChange::effect()
post : self.productAttribute.increment = self.newIncrement **and**
self.productAttribute.sign = self.newSign

EditDownloadableAttribute



context EditDownloadableProductAttribute::effect()
post :
self.downloadable.filename = self.newFilename **and**
self.downloadable.expiryDays = self.newExpiryDays **and**
self.downloadable.maximumDownloadCount = self.newMaximumDonwloadCount

ProductAttributeStatusChange



context ProductAttributeStatusChange::effect()
post : self.productAttribute.status = self.newStatus

Example test programs

```
testprogram ProductOptionsManagement{
        catalan := new Language(name:='Catalan', code:='CAT');
        english := new Language(name:='English', code:='EN');

        fixturecomponent optionShirtSizeInitialized{
                shirtSize:=new Option;
                englishName:=new StringDT(string:='Shirt size');
                catalanName:=new StringDT(string:='Mida de samarretes');
                new HasOptionName
                   (option:=shirtSize, optionName:=englishName, optionLanguage:=english);
                new HasOptionName
                   (option:=shirtSize, optionName:=catalanName, optionLanguage:=catalan);
        }

        fixturecomponent valueSmallInitialized{
                small:=new Value;
                englishName:=new StringDT(string:='Small');
                catalanName:=new StringDT(string:='Petit');
                new HasValueName(value:=small,
                                 valueName:=englishName, valueLanguage:=english);
                new HasValueName(value:=small,
                                 valueName:=catalanName, valueLanguage:=catalan);
        }

        test NewProductOptionWithoutNamesForSomeLanguages{
                //We should specify the product option name in each language
                s:=new StringDT(string:='Size');
                npo:=new NewProductOption;
                new HasNewOptionName(nameOfOption:=s,
                          languageOfOption:=english,productOptionNameEvent:=this));
                npo may not occur;
        }

        test NewProductOptionsWithAllNamesSpecified{
                //We test a valid invocation of the event
                englishName:=new StringDT(string:='Size');
                catalanName:=new StringDT(string:='Mida');
                npo:=new NewProductOption;
                new HasNewOptionName(nameOfOption:=catalanName,
                          languageOfOption:=catalan,productOptionNameEvent:=this);
                new HasNewOptionName(nameOfOption:=englishName,
                          languageOfOption:=english,productOptionNameEvent:=this);
                npo occurs;
        }

        test NewProductOptionThatAlreadyExists{
                load optionShirtSizeInitialized;
                differentName:=new StringDT(string:='AnyName');
                npo:=new NewProductOption;
                new HasNewOptionName(nameOfOption:=catalanName,
                           languageOfOption:=catalan,productOptionNameEvent:=this);
                new HasNewOptionName(nameOfOption:=differentName,
                          languageOfOption:=english,productOptionNameEvent:=this));
                npo may not occur;

                npo2:=new NewProductOption;
                new HasNewOptionName(nameOfOption:=differentName,
                          languageOfOption:=catalan,productOptionNameEvent:=this);
                new HasNewOptionName(nameOfOption:=englishName,
                          languageOfOption:=english,productOptionNameEvent:=this));
                npo2 may not occur;

                npo3:=new NewProductOption;
                new HasNewOptionName(nameOfOption:=catalanName,
                          languageOfOption:=catalan,productOptionNameEvent:=this);
                new HasNewOptionName(nameOfOption:=englishName,
                           languageOfOption:=english,productOptionNameEvent:=this));
```

```
                npo3 may not occur;
        }

        test EditProductOptionWithoutNamesForSomeLanguages{
                load optionShirtSizeInitialized;
                s:=new StringDT(string:='Size');
                npo:=new EditProductOption(option:=shirtSize);
                new HasNewOptionName(nameOfOption:=s,languageOfOption:=english,
                        productOptionNameEvent:=this))
                npo may not occur;
        }
        test EditProductOptionsWithAllNamesSpecified{
                load optionShirtSizeInitialized;
                englishName:=new StringDT(string:='Size');
                catalanName:=new StringDT(string:='Mida');
                epo:=new EditProductOption(option:=shirtSize);
                new HasNewOptionName(nameOfOption:=catalanName,
                        languageOfOption:=catalan,productOptionNameEvent:=this);
                new HasNewOptionName(nameOfOption:=englishName,
                        languageOfOption:=english,productOptionNameEvent:=this));
                epo occurs;
        }

        test UnapplicableProductOptionEdition{
                load optionShirtSizeInitialized;
                //We add to the IB another option
                sleeveType:=new Option;
                englishName:=new StringDT(string:='Sleeve type');
                catalanName:=new StringDT(string:='Tipus de maniga');
                new HasOptionName(option:=sleeveType,
                        optionName:=englishName, optionLanguage:=english);
                new HasOptionName(option:=sleeveType,
                        optionName:=catalanName, optionLanguage:=catalan);
                check consistency;
                differentName:=new StringDT(string:='AnyName');
                epo:=new EditProductOption(option:=shirtSize);
                new HasNewOptionName(nameOfOption:=catalanName,
                        languageOfOption:=catalan,productOptionNameEvent:=this);
                new HasNewOptionName(nameOfOption:=differentName,
                        languageOfOption:=english,productOptionNameEvent:=this));
                epo may not occur;
                epo2:= new EditProductOption(option:=shirtSize);
                new HasNewOptionName(nameOfOption:=differentName,
                        languageOfOption:=catalan,productOptionNameEvent:=this);
                new HasNewOptionName(nameOfOption:=englishName,
                        languageOfOption:=english,productOptionNameEvent:=this));
                epo2 may not occur;
                epo3:=new EditProductOption(option:=shirtSize);
                new HasNewOptionName(nameOfOption:=catalanName,
                        languageOfOption:=catalan,productOptionNameEvent:=this);
                new HasNewOptionName(nameOfOption:=englishName,
                        languageOfOption:=english,productOptionNameEvent:=this));
                epo3 may not occur;
        }
}
testprogram DeleteProductOptions{

        shoesSize:=new Option;
        shirtSize:=new Option;
        small:=new Value;

        test deleteOptionWithoutValues{
                new DeleteProductOption(option:=shirtSize);
        }

        test deleteOptionThatIsPartOfAProductAttribute{
                barcelonaTShirt:=new Product;
                smallShirt:=new Attribute(option:=shirtSize,value:=small);
                new ProductAttribute(product:=barcelonaTShirt,attribute:=smallShirt);
                new DeleteProductOption(option:=shirtSize) may not occur;
        }
```

```
        test deleteOptionWithAssociatedValuesNotUsedInOtherOptions{
                new Attribute(option:=shirtSize,value:=small);
                new DeleteProductOption(option:=shirtSize) occurs;
                assert true Value.allInstances->excludes(small);
        }

        test deleteOptionWithAssociatedValuesUsedInOtherOptions{
                new Attribute(option:=shirtSize,value:=small);
                new Attribute(option:=shoesSize,value:=small);
                new DeleteProductOption(option:=shirtSize) occurs;
                assert true Value.allInstances->includes(small);
        }
}
```

```
testprogram ProductOptionsValuesManagement{

        catalan := new Language(name:='Catalan', code:='CAT');
        english := new Language(name:='English', code:='EN');

        shirtSize:=new Option;
        englishName:=new StringDT(string:='Shirt size');
        catalanName:=new StringDT(string:='Mida de samarretes');
        new HasOptionName(option:=shirtSize,
                          optionName:=englishName, optionLanguage:=english);
        new HasOptionName(option:=shirtSize,
                          optionName:=catalanName, optionLanguage:=catalan);

        fixturecomponent valueSmallInitialized{
                smallInEnglish:=new StringDT(string:='Small');
                smallInCatalan:=new StringDT(string:='Petit');
                small:=new Value;
                new HasValueName(value:=small,
                                 valueName:=smallInEnglish, valueLanguage:=english);
                new HasValueName(value:=small,
                                 valueName:=smallInCatalan, valueLanguage:=catalan);
        }

        test NewProductOptionValueWithoutNamesForSomeLanguages{
                //We should specify the product option name in each language and an option
                smallInEnglish:=new StringDT(string:='Small');
                npov:=new NewProductOptionValue;
                new HasNewValueName(nameOfValue:=smallInEnglish,
                      languageOfValue:=english,productValueNameEvent:=this));
                npov may not occur;
        }

        test NewProductOptionValueWithAllNamesSpecified{
                //We test a valid invocation of the event
                smallInEnglish:=new StringDT(string:='Small');
                smallInCatalan:=new StringDT(string:='Petit');
                npov:=new NewProductOptionValue(option:=shirtSize);
                new HasNewValueName(nameOfValue:=smallInEnglish,
                       languageOfValue:=english,productValueNameEvent:=this);
                new HasNewValueName(nameOfValue:=smallInCatalan,
                     languageOfValue:=catalan,productValueNameEvent:=this))
                npov occurs;
        }

        test NewProductOptionValueThatAlreadyExists{
                //IB state with a product option value
                load valueSmallInitialized;
                smallInEnglish:=new StringDT(string:='Small');
                smallInCatalan:=new StringDT(string:='Petit');
                //The creation of a product option value with the same name in at least one
               //language should not occur
                differentName:=new StringDT(string:='AnyName');
                npov1:=new NewProductOptionValue(option:=shirtSize);
                new HasNewValueName(nameOfValue:=smallInCatalan,
                          languageOfValue:=catalan,productValueNameEvent:=this);
```

```
                new HasNewValueName(nameOfValue:=differentName,
                          languageOfValue:=english,productValueNameEvent:=this));
                npov1 may not occur;
                npov2:=new NewProductOptionValue(option:=shirtSize);
                new HasNewValueName(nameOfValue:=differentName,
                          languageOfValue:=catalan,productValueNameEvent:=this);
                new HasNewValueName(nameOfValue:=smallInEnglish,
                          languageOfValue:=english,productValueNameEvent:=this));
                npov2 may not occur;
                npov3:=new NewProductOptionValue(option:=shirtSize);
                new HasNewValueName(nameOfValue:=smallInCatalan,
                          languageOfValue:=catalan,productValueNameEvent:=this);
                new HasNewValueName(nameOfValue:=smallInEnglish,
                          languageOfValue:=english,productValueNameEvent:=this));
               npov3 may not occur;
        }

        test EditProductOptionValueWithoutNamesForSomeLanguages{
                //We should specify the product Value name in each language
                load valueSmallInitialized;
                s:=new StringDT(string:='Small');
                epov:=new EditProductOptionValue(option:=shirtSize, value:=small);
                new HasNewValueName(nameOfValue:=s,
                          languageOfValue:=english,productValueNameEvent:=this));
                epov may not occur;
        }

        test EditProductValuesWithAllNamesSpecified{
                load valueSmallInitialized;
                smallInEnglish:=new StringDT(string:='Small');
                smallInCatalan:=new StringDT(string:='Petit');
                //We test a valid invocation of the event
                epov:=new EditProductOptionValue(option:=shirtSize,value:=small);
                new HasNewValueName(nameOfValue:=smallInCatalan,
                          languageOfValue:=catalan,productValueNameEvent:=this);
                new HasNewValueName(nameOfValue:=smallInEnglish,
                          languageOfValue:=english,productValueNameEvent:=this));
                epov occurs;
        }

        test UnapplicableProductValueEdition{
                load valueSmallInitialized;
                //We add to the IB another Value
                large:=new Value;
                englishName:=new StringDT(string:='Large');
                catalanName:=new StringDT(string:='Gran');
                new HasValueName(value:=large,
                                  valueName:=englishName, valueLanguage:=english);
                new HasValueName(value:=large,
                                  valueName:=catalanName, valueLanguage:=catalan);
                check consistency;
                differentName:=new StringDT(string:='AnyName');
                new EditProductOptionValue(value:=small,option:=shirtSize);
                new HasNewValueName(nameOfValue:=catalanName,languageOfValue:=catalan,
                                  productValueNameEvent:=this);
                new HasNewValueName(nameOfValue:=differentName,languageOfValue:=english,
                                  productValueNameEvent:=this)) may not occur;
                epov:=new EditProductOptionValue(value:=small,option:=shirtSize);
                new HasNewValueName(nameOfValue:=differentName,
                                  languageOfValue:=catalan,productValueNameEvent:=this);
                new HasNewValueName(nameOfValue:=englishName,
                                  languageOfValue:=english,productValueNameEvent:=this));
                epov may not occur;
                epov:=new EditProductOptionValue(value:=small,option:=shirtSize);
                new HasNewValueName(nameOfValue:=catalanName,
                           languageOfValue:=catalan,productValueNameEvent:=this);
                new HasNewValueName(nameOfValue:=englishName,
                           languageOfValue:=english,productValueNameEvent:=this));
                epov may not occur;
        }
}
```

```
testprogram DeleteProductOptionsValues{
        shoesSize:=new Option;
        shirtSize:=new Option;
        small:=new Value;

        fixturecomponent barcelonaTShirtInitialized{
                barcelonaTShirt:=new Product;
                smallShirt:=new Attribute(option:=shirtSize,value:=small);
                barcelonaSmallTShirt:=new ProductAttribute
                        (product:=barcelonaTShirt,attribute:=smallShirt);
        }

        test deleteValueNotUsed{
                new DeleteProductOptionValue(value:=small) occurs;
        }

        test deleteValueOfTwoOptions{
                small.option:=shoesSize,shirtSize;
                new DeleteProductOptionValue(value:=small) occurs;
        }

        test deleteValueThatIsPartOfAProductAttribute{
                load barcelonaTShirtInitialized;
                new DeleteProductOptionValue(value:=small) may not occur;
        }

        test deleteValueThatIsPartOfAnOrder{
                load barcelonaTShirtInitialized;
                //We create an order
                o:= new Order;
                ol:= new OrderLine(product:=barcelonaTShirt,order:=o);
                euro:=new Currency;
                o.currency:=euro;
                dos:=new OrderStatus;
                osc := new OrderStatusChange(order:=o,orderStatus:=dos);
                sm:= new FlatRate(status:=#enabled);
                pm:= new Nochex(status:=#enabled);
                o.shippingMethod:=sm;
                o.paymentMethod:=pm;
                usa:=new Country;
                a:= new Address(country:=usa);
                c := new Customer(address:=a,primary:=a);
                o.customer:=c;
                ola:=new OrderLineAttribute(attribute:=smallShirt, orderLine:=ol);
                //We cannot delete a value wich is part of an attribute of an order...
                new DeleteProductOptionValue(value:=small) may not occur;
                delete barcelonaSmallTShirt;
                check consistency;
                //...although the product attribute is not offered
                new DeleteProductOptionValue(value:=small) may not occur;
        }
}
```

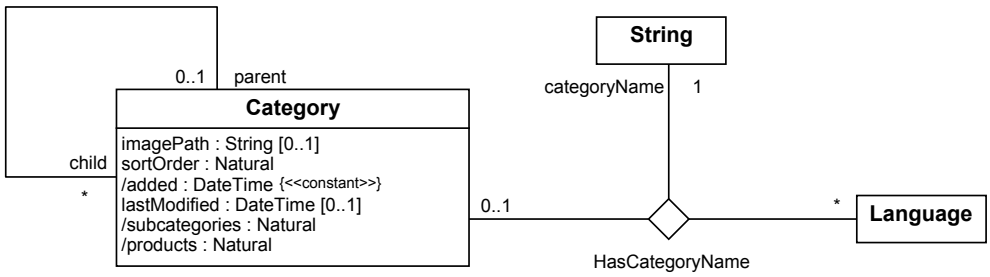
```
testprogram ProductOptionsManagement{
        edition:=new Option; version:=new Option;
        special:=new Value;
        specialWithDirectorComments:=new Value;
        catalan:=new Value;
        vickyCristinaBarcelonaDVD:=new Product(netPrice:=20);
        specialEdition:=new Attribute(option:=edition,value:=special);
        specialWithCommentsEdition:=new Attribute
                        (option:=edition,value:=specialWithDirectorComments);
        catalanVersion:=new Attribute(option:=version,value:=catalan);

        fixturecomponent vickyCristinaBarcelonaSpecialDVDEditionInitialize{
                vcbSpecialDVDEdition:=new ProductAttribute
                        (product:=vickyCristinaBarcelonaDVD, attribute:=specialEdition);
                vcbSpecialDVDEdition.increment:=3;
                vcbSpecialDVDEdition.sign:=#plus;
        }
```

```
        test NewProductAttributeWithValidOptionValuePair{
                new NewProductAttribute
                      (product:=vickyCristinaBarcelonaDVD,option:=edition,
                       value:=special, increment:=3,sign:=#plus) occurs;
        }

        test NewProductAttributeWithInvalidOptionValuePair{
                new NewProductAttribute(product:=vickyCristinaBarcelonaDVD,
                       option:=edition, value:=catalan,
                       increment:=3,sign:=#plus) may not occur;
        }

        test NewProductAttributeThatAlreadyExists{
                load vickyCristinaBarcelonaSpecialDVDEditionInitialize;
                //If a product attribute with the same option and value already exists in the
                //IB, the event NewProduct Attribute should not occur
                new NewProductAttribute(product:=vickyCristinaBarcelonaDVD, option:=edition,
                       value:=special,increment:=5,sign:=#minus) may not occur;
        }

        test EditProductAttribute{
                load vickyCristinaBarcelonaSpecialDVDEditionInitialize;
                new AttributeChange
                       (productAttribute:=vcbSpecialDVDEdition,
                        newValue:=specialWithDirectorComments, newOption:=edition) occurs;
        }

        test EditIncrementAndSign{
                load vickyCristinaBarcelonaSpecialDVDEditionInitialize;
                new IncrementAndSignAttributeChange(productAttribute:=vcbSpecialDVDEdition,
                       newIncrement:=5,newSign:=#plus) occurs;
        }

        test InvalidEditProductAttribute{
                load vickyCristinaBarcelonaSpecialDVDEditionInitialize;
                vcbCatalanVersion:=new ProductAttribute(product:=vickyCristinaBarcelonaDVD,
                        attribute:=catalanVersion);
                new AttributeChange(productAttribute:=vcbCatalanVersion,
                        newValue:=catalan, newOption:=edition) may not occur;
        }

        test DeleteProductAttributeNotUsedInAnyOrder{
                load vickyCristinaBarcelonaSpecialDVDEditionInitialize;
                new DeleteProductAttribute(productAttribute:=vcbSpecialDVDEdition) occurs;
                assert true ProductAttribute.allInstances->size()=0;
        }

        test DeleteProductAttributUsedInAnOrder{
                load vickyCristinaBarcelonaSpecialDVDEditionInitialize;
                //We create an order
                o:= new Order;
                ol:= new OrderLine(product:=vickyCristinaBarcelonaDVD,order:=o);
                euro:=new Currency;
                o.currency:=euro;
                dos:=new OrderStatus;
                osc := new OrderStatusChange(order:=o,orderStatus:=dos);
                sm:= new FlatRate(status:=#enabled);
                pm:= new Nochex(status:=#enabled);
                o.shippingMethod:=sm;
                o.paymentMethod:=pm;
                spain:=new Country;
                a:= new Address(country:=spain);
                c := new Customer(address:=a,primary:=a);
                o.customer:=c;
                ola:=new OrderLineAttribute(attribute:=specialEdition, orderLine:=ol);
                new DeleteProductAttribute(productAttribute:=vcbSpecialDVDEdition) occurs;
                assert true ProductAttribute.allInstances->includes(vcbSpecialDVDEdition);
                assert equals vcbSpecialDVDEdition.status #disabled;
        }
}
```

9.11. Product categories

Structural schema

Products are grouped into categories which are arranged hierarchically.



context Category **def:**
allParents() : Set(Category) = self.parent -> union(self.parent.allParents())

[DR1] *Category::**added*** is the *DateTime* of category creation.

context Category::added():DateTime
body : Now()

[DR2] *Category::**subcategories*** is the number of subcategories owned by the category.

context Category::subcategories(): Natural
body : self.child -> size()

[DR3] *Category::**products*** is the number of products owned by the category.

context Category::products(): Natural
body : Category.allInstances() -> select(c | c.allParents() -> includes(self)) ->union(Set{self}).product -> size()

[IC1] In each language, each category has a unique name.

context Language::categoryNameIsUnique(): Boolean
body : self.hasCategoryName -> isUnique(name)

[IC2] There are no cycles in category hierarchy.

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

Use cases

Add a product category

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a category.

Main Success Scenario:

1. The store administrator provides the details of the new product category, including its parent category, if any:
   [→*NewCategory*]
2. The system validates that the data is correct.
3. The system saves the new category.

Edit a product category

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a category.

Main Success Scenario:

1. The store administrator selects the category to be edited.
2. The store administrator provides the new details of the selected category:
   [→*EditCategory*]
3. The system validates that the data is correct.
4. The system saves the changes.

Move a product category

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the placement of a category in the category hierarchy.

Main Success Scenario:

1. The store administrator selects the category to be moved.

2. The store administrator indicates the new parent category, if any:

   [→*MoveCategory*]

3. The system validates that the data is correct.
4. The system saves the new placement.

Delete a product category

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a category.

Main Success Scenario:

1. The store administrator selects the category to be deleted.
2. The system warns the store administrator of the number of subcategories and products linked to the category to be deleted.
3. The store administrator confirms that he wants to delete the category:

   [→*DeleteCategory*]

4. The system deletes the selected category and its subcategories. The products linked to the deleted category or its subcategories are removed from the system if they do not participate in any orders. The system changes the status of the products which participate in orders to out of stock.

Move a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the category of a product.

Main Success Scenario:

1. The store administrator selects the product to be moved.
2. The store administrator indicates the new category of the selected product, if any:

   [→*MoveProduct*]

3. The system validates that the data is correct.
4. The system saves the new placement.

Link a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to link a product to another category.

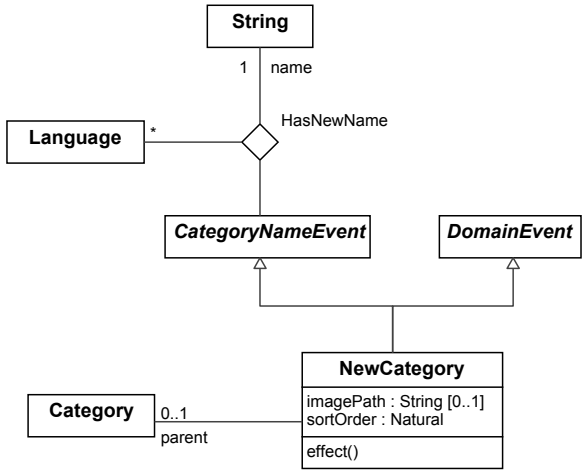
Main Success Scenario:

1. The store administrator selects the product to be linked.
2. The store administrator indicates the new category of the selected product, if any :

   [→*LinkProduct*]
3. The system links the product.

Events

NewCategory



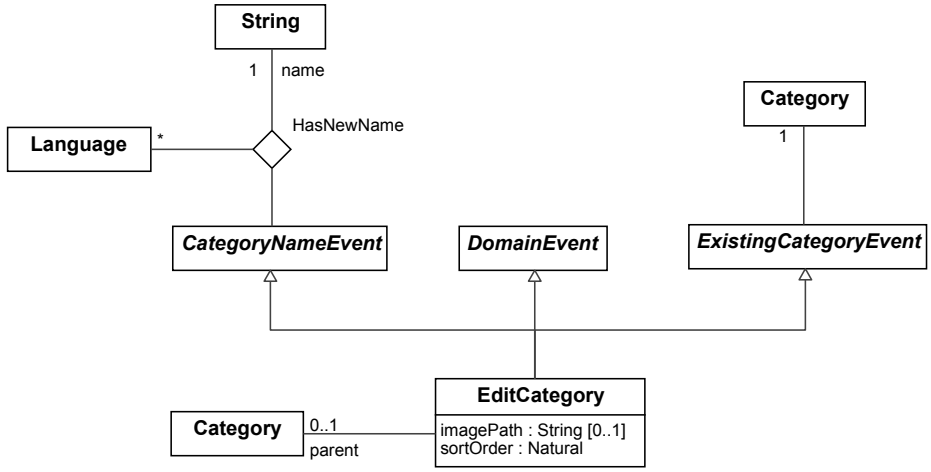
«IniIC»

```
context NewCategory::categoryDoesNotExist(): Boolean
  body :  Language.allInstances() -> forAll ( l |
              l.hasCategoryName.categoryName ->
              excludes(self.hasNewName->select(language=l)->any(true).name))
```

```
context  NewCategory::effect()
  post :
     c.oclIsNew()  and
     c.oclIsTypeOf(Category) and
     c.imagePath = self.imagePath and
```

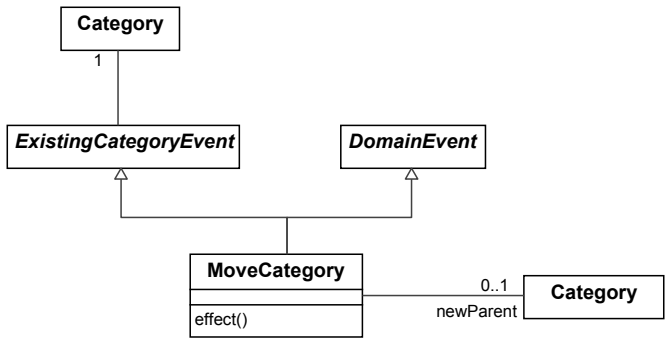
```
    c.sortOrder = self.sortOrder and
    c.parent = self.parent and
    Language.allInstances() ->
       forAll (l| self.hasNewName -> select(language=l).name =
                  c.hasCategoryName->select(language=l).categoryName)
```

EditCategory




```
«IniIC»
context EditCategory::categoryDoesNotExist():Boolean
  body:  Language.allInstances -> forAll ( l |
         l.hasCategoryName.categoryName.string
         -> excludes(self.hasNewName -> any(language=l).name) or
         (self.hasNewName->any(language=l).name =
         self.category.hasCategoryName->any(language=l).categoryName))

«IniIC»
context EditCategory::cyclesDoNotAppear():Boolean
            self.category.allParents()->union(Set{self.newParent})->excludes(self.category)

context  EditCategory::effect()
  post :
     self.category.imagePath = self.imagePath and
     self.category.sortOrder = self.sortOrder and
     self.category.parent = self.parent and
     Language.allInstances() ->
        forAll(l|

           self.hasNewName->select(language=l)->any(true).name=
           self.category.hasCategoryName->select(language=l).categoryName)
  post :
     self.category.lastModified = Now()
```

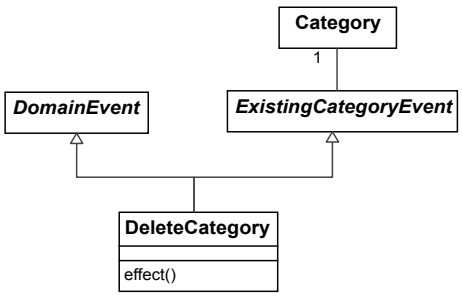
MoveCategory



«IniIC»
context MoveCategory::cyclesDoNotAppear():Boolean
self.newParent.allParents()->excludes(self.category)

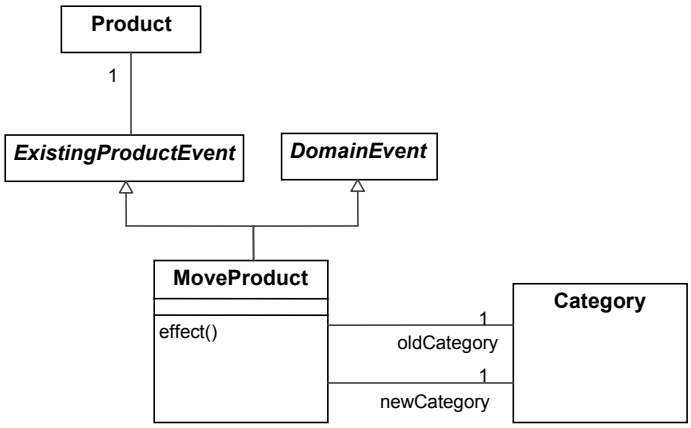
context MoveCategory::effect()
post : self.category.parent = self.newParent

DeleteCategory



context DeleteCategory::effect()
post deleteCategoryAndSubcategories**:**
Category.allInstances->excludes(self.category@pre) **and**
self.allChilds(category@pre) -> forAll(c | Category.allInstances->excludes(c))
post deleteProductsOfCategory**:**
self.category@pre.product@pre -> forAll(p |
if p.orderLine -> notEmpty() **then** p.status = ProductStatus::outOfStock
else p@pre.oclIsKindOf(OclAny)
endif)
post deleteProductsOfChildCategory**:**
self.category@pre.child@pre.product@pre -> forAll(p |
if p.orderLine -> notEmpty() **then** p.status = ProductStatus::outOfStock
else p.oclIsKindOf(OclAny)
endif)

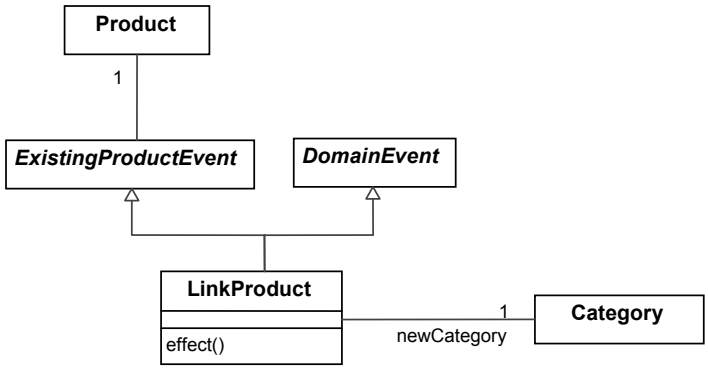
MoveProduct



«IniIC»
context MoveProduct::oldCategoryIsValid(): Boolean
body: product.category->includes(self.oldCategory)

context MoveProduct::effect()
post: self.product.category -> includes(self.newCategory) **and**
self.product.category -> excludes(self.oldCategory)

LinkProduct



context LinkProduct::effect()
post: self.product.category -> includes(self.newCategory)

Example test programs

```
testprogram ProductCategoriesManagement{
        //Test cases are based on a multilingual online shop with two languages
        italian := new Language(name:='Italian', code:='IT');
        english := new Language(name:='English', code:='EN');

        fixturecomponent woodenToysCategoryInitialized{
                woodenToysInEnglish:=new StringDT(string:='Wooden toys');
                woodenToysInItalian:=new StringDT(string:='Giocattoli di legno');
                woodenToys:=new Category;
                new HasCategoryName(category:=woodenToys, categoryName:=woodenToysInEnglish,
                                language:=english);
                new HasCategoryName(category:=woodenToys, categoryName:=woodenToysInItalian,
                                language:=italian);
        }

        fixturecomponent gamesCategoryInitialized{
                gamesInEnglish:=new StringDT(string:='Games');
                gamesInItalian:=new StringDT(string:='Giocci di societa');
                games:=new Category;
                new HasCategoryName(category:=games, categoryName:=gamesInEnglish,
                                language:=english);
                new HasCategoryName(category:=games, categoryName:=gamesInItalian,
                                language:=italian);
        }

        test NewCategory{
                //We should specify the product option name in each language and an option
                gamesInEnglish:=new StringDT(string:='Games');
                gamesInItalian:=new StringDT(string:='Giocci di societa');
                nc:=new NewCategory;
                new HasNewName(name:=gamesInEnglish,
                                languageOfCategory:=english,categoryNameEvent:=this);
                new HasNewName(name:=gamesInItalian,languageOfCategory:=italian,
                                categoryNameEvent:=this))
                nc occurs;
        }

        test NewSubcategory{
                load woodenToysCategoryInitialized;
                //We should specify the product option name in each language and an option
                trainsInEnglish:=new StringDT(string:='Trains');
                trainsInItalian:=new StringDT(string:='Trenini');
                nc:=new NewCategory(parent:=woodenToys);
                new HasNewName(name:=trainsInEnglish,languageOfCategory:=english,
                                 categoryNameEvent:=this);
                new    HasNewName(name:=trainsInItalian,languageOfCategory:=italian,
                                 categoryNameEvent:=this));
                nc occurs;
        }

        test EditCategory{
                load woodenToysCategoryInitialized;
                trainsInEnglish:=new StringDT(string:='Trains');
                trainsInItalian:=new StringDT(string:='Trenini');
                nc:=new NewCategory(parent:=woodenToys);
                new HasNewName(name:=trainsInEnglish,languageOfCategory:=english,
                                 categoryNameEvent:=this);
                new HasNewName(name:=trainsInItalian,languageOfCategory:=italian,
                                 categoryNameEvent:=this));
                nc occurs;
                ec:=new EditCategory(category:=woodenToys);
                new HasNewName(name:=trainsInEnglish,languageOfCategory:=english,
                                 categoryNameEvent:=this);
                new HasNewName(name:=woodenToysInItalian,languageOfCategory:=italian,
                                 categoryNameEvent:=this))
                ec may not occur;
        }
```

```
        test EditCategoryCausingACycle{
                load woodenToysCategoryInitialized;
                woodenToysInEnglish:=new StringDT(string:='Wooden toys');
                woodenToysInItalian:=new StringDT(string:='Giocattoli di legno');
                ed:=new EditCategory(category:=woodenToys,newParent:=woodenToys);
                new HasNewName(name:=woodenToysInEnglish,languageOfCategory:=english,
                                categoryNameEvent:=this);
                new HasNewName(name:=woodenToysInItalian,languageOfCategory:=italian,
                                 categoryNameEvent:=this));
                ec may not occur;
        }

        test MoveCategory{
                load woodenToysCategoryInitialized;
                load gamesCategoryInitialized;
                new MoveCategory(category:=games,newParent:=woodenToys) occurs;
                assert equals games.parent woodenToys;
        }

        test MoveCategoryCausingCycles{
                load woodenToysCategoryInitialized;
                load gamesCategoryInitialized;
                games.parent:=woodenToys;
                trainsInEnglish:=new StringDT(string:='Trains');
                trainsInItalian:=new StringDT(string:='Trenini');
                nc:=new NewCategory(parent:=games);
                new HasNewName(name:=trainsInEnglish,languageOfCategory:=english,
                                categoryNameEvent:=this);
                new HasNewName(name:=trainsInItalian,languageOfCategory:=italian,
                                categoryNameEvent:=this));
                nc occurs;
                trains := HasCategoryName.allInstances
                                ->any(categoryName=trainsInEnglish).category;
                new MoveCategory(category:=woodenToys,newParent:=trains) may not occur;
        }

        test DeleteCategoryWithoutSubcategories{
                load woodenToysCategoryInitialized;
                new DeleteCategory(category:=woodenToys) occurs;
        }

        test DeleteCategoryWithSubcategories{
                load woodenToysCategoryInitialized;
                load gamesCategoryInitialized;
                new MoveCategory(category:=games,newParent:=woodenToys) occurs;
                new DeleteCategory(category:=woodenToys) occurs;
                assert true Category.allInstances->excludes(woodenToys);
                assert true Category.allInstances->excludes(games);
        }
}
```

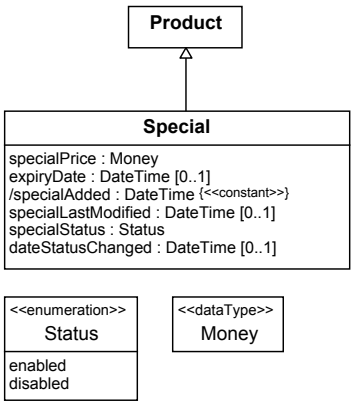
```
testprogram ProductMovementsInCategories{

        p := new Product;
        c1 := new Category;
        c2 := new Category;
        c3 := new Category;

        test MoveBetweenCategories{
                p.category:=c1;
                new MoveProduct(product:=p, oldCategory:=c1, newCategory:=c2) occurs;
                assert equals p.category Set{c2};
        }

        test InvalidMoveBetweenCategories{
                new MoveProduct(product:=p, oldCategory:=c1, newCategory:=c2) may not occur;
        }
```

```
        test LinkProduct{
              /*Link a product makes possible to assign a product
                to more than one categories
                LinkProduct add categories of a product
                preserving the already assigned categories*/
                p.category:=c1;
                new LinkProduct(product:=p, newCategory:=c2) occurs;
                assert equals p.category Set{c2,c1};
        }

        test SubcategoriesAndProductsDerivedInformation{
                //We add two new products to the IB
                p2:=new Product;
                p3:=new Product;

                //We establish the categories hierarchy
                c1.child:=c2,c3;
                //We organize products
                c1.product:=p;
                c2.product:=p2,p3;
                //We materialize the derived attributes
                c1._subcategories:=2;
                c2._subcategories:=0;
                c3._subcategories:=0;
                c1._products:=3;
                c2._products:=2;
                c3._products:=0;
                check consistency;
        }
}
```

9.12. Specials

Structural schema

osCommerce allows offering specials. That is, lower prices for a set of products can be offered during a specific time period.



[DR1] *Special::**added*** is the *DateTime* when the special was created

context Special::added():DateTime
body : Now()

Add a special

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a special.

Main Success Scenario:

1. The store administrator selects the product which will be offered in a special price.
2. The store administrator provides the details of the special:

   [→*NewSpecial*]
3. The system validates that the data is correct.
4. The system saves the new special.

Edit a special

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a special.

Main Success Scenario:

1. The store administrator selects the special to be edited.
2. The store administrator provides the new details of the selected special:

   [→*EditSpecial*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a special

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a special.

Main Success Scenario:

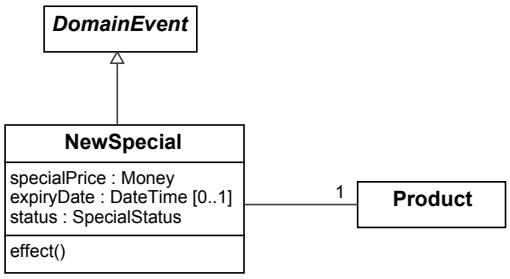
1. The store administrator selects the special to be deleted.
2. The system asks for the confirmation of the store administrator.

3. The store administrator confirms that he wants to delete the special:

   [→*DeleteSpecial*]

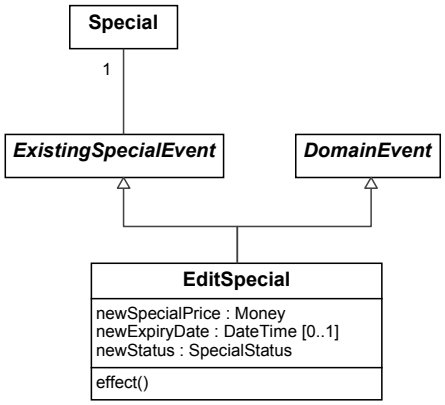
4. The system deletes the special.

NewSpecial



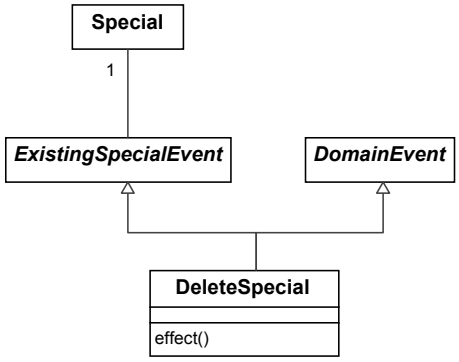
context NewSpecial::effect()
  post :
    self.product.oclIsTypeOf(Special) **and**
    self.product.oclAsTypeOf(Special).specialPrice=self.specialPrice **and**
    self.product.oclAsTypeOf(Special).expiryDate=self.expiryDate **and**
    self.product.oclAsTypeOf(Special).status=self.status

EditSpecial



context EditSpecial::effect()
  post :
    self.special.specialPrice = self.newSpecialPrice **and**
    self.special.expiryDate = self.newExpiryDate **and**
    self.special.status = self.newStatus
  post :
    self.special.lastModified = Now()
  post :
    self.special@pre.status <> self.newStatus **implies**
    self.special.dateStatusChanged = Now()

DeleteSpecial




```
context DeleteSpecial::effect()
  post :
          Special.allInstances()->excludes(special@pre)  and
          (Product.allInstances() - Product.allInstances()@pre) -> one(p:Product |
                  p.status = special@pre.status@pre and
                  p.available = special@pre.available@pre and
                  p.netPrice = special@pre.netPrice@pre and
                  p.quantityOnHand = special@pre.quantityOnHand@pre and
                  p.model = special@pre.model@pre and
                  p.imagePath = special@pre.imagePath@pre and
                  p.weight = special@pre.weight@pre and
                          p.category = special@pre.category@pre and
                  p.manufacturer = special@pre.manufacturer@pre and
                  p.taxClass = special@pre.taxClass@pre and
                  p.lastModified=Now() and
                  Language.allInstances ->
                          forAll (l|
                             special@pre.productInLanguage->select(language=l).name =
                             p.productInLanguage->select(language=l).name))
```

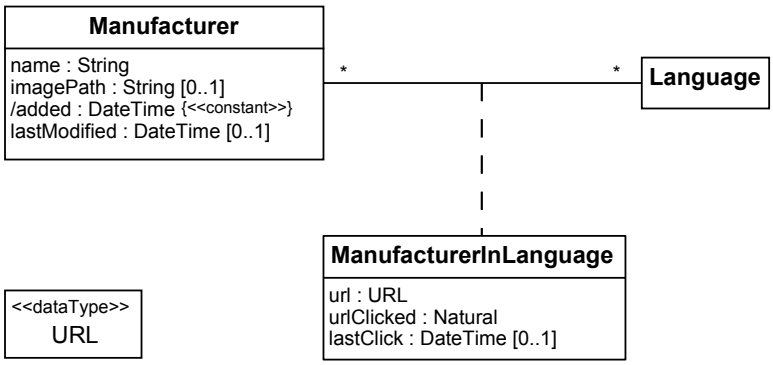
Example test program

```
testprogram SpecialsManagement{

        skypePhone:=new Product(netPrice:=90);

        test AddEditAndDeleteSpecials{
                ns:=new NewSpecial(product:=skypePhone,specialPrice:=60,status:=#disabled)
                    occurs;
                assert true ns.product.specialNetPrice().isUndefined();
                new EditSpecial(special:=ns.product,newSpecialPrice:=60,newStatus:=#enabled)
                    occurs;
                assert equals ns.product.specialNetPrice() 60;
                new EditSpecial(special:=ns.product,newSpecialPrice:=55,newStatus:=#enabled)
                    occurs;
                assert equals ns.product.specialNetPrice() 55;
                specialProduct:=ns.product;
                new DeleteSpecial(special:=specialProduct) occurs;
                assert true ns.product.specialNetPrice().isUndefined();
        }
}
```

9.13. Manufacturers

Structural schema

In *osCommerce*, the products in the store are manufactured by manufacturers.



[DR1] *Manufacturer::**added*** is the *DateTime* when the *Manufacturer* was created.

```
context Manufacturer::added():DateTime
 body :  Now()
```

[IC1] A manufacturer is identified by its name

```
context Manufacturer::nameIsUnique(): Boolean
  body : Manufacturer.allInstances() -> isUnique(name)
```

[IC2] Each manufacturer must have a URL in each language

```
context Manufacturer::aURLInEachLanguage(): Boolean
  body : self.language ->size() = Language.allInstances() -> size()
```

Use cases

Add a manufacturer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a manufacturer.

Main Success Scenario:

1. The store administrator provides the details of the new manufacturer:

   [→*NewManufacturer*]

2. The system validates that the data is correct.
3. The system saves the new manufacturer.

Edit a manufacturer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a manufacturer.

Main Success Scenario:

1. The store administrator selects the manufacturer to be edited.
2. The store administrator provides the new details of the selected manufacturer:

   [→*EditManufacturer*]

3. The system validates that the data is correct.
4. The system saves the changes.

Delete a manufacturer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a manufacturer.

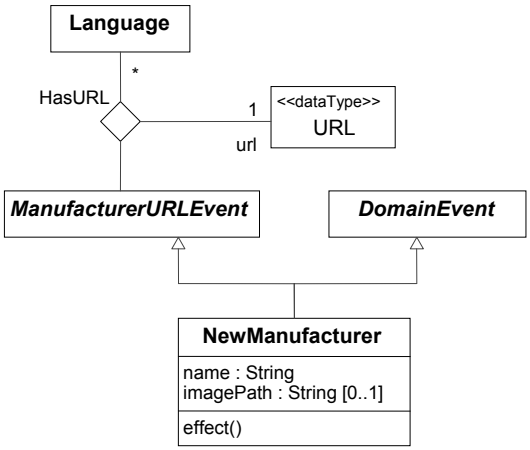
Main Success Scenario:

1. The store administrator selects the manufacturer to delete.
2. The system warns the store administrator of the number of products linked to the manufacturer to be deleted.
3. The store administrator confirms that he wants to delete the manufacturer:

   [→*DeleteManufacturer*]

4. The system deletes the manufacturer and, if requested, changes the status of the products manufactured by it to out of stock.

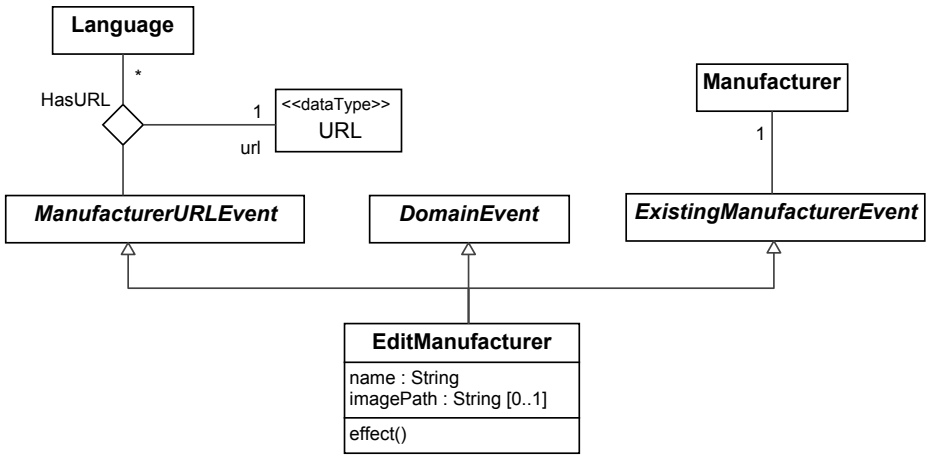
NewManufacturer



«IniIC»
context NewManufacturer::manufacturerDoesNotExist(): Boolean
body :
not Manufacturer.allInstances() -> exists (m | m.name=self.name)

context NewManufacturer::effect()
post :
m.oclIsNew() **and**
m.oclIsTypeOf(Manufacturer) **and**
m.name = self.name **and**
m.imagePath = self.imagePath **and**
Language.allInstances() ->
forAll (l|
self.hasURL -> select(language=l).url =
m.manufacturerInLanguage->select(language=l).url)

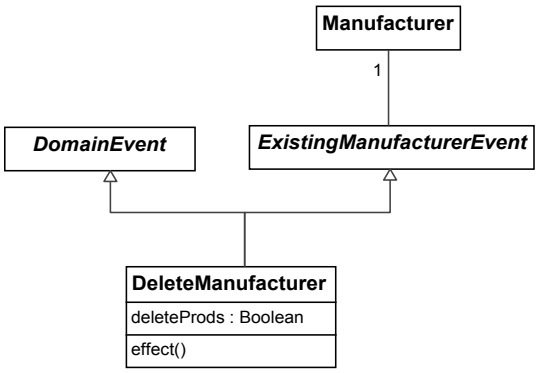
EditManufacturer



«IniIC»
context EditManufacturer::manufacturerDoesNotExist(): Boolean
body :
(Manufacturer.allInstances() –Set{self.manufacturer}).name-> excludes(self.name)

```
context  EditManufacturer::effect()
  post :
    self.manufacturer.name = self.name and
    self.manufacturer.imagePath = self.imagePath and
    Langauge.allInstances() ->
      forAll(l|
        self.hasURL->select(language=l).url=
        self.manufacturer.manufacturerInLanguage->
          select(language=l).url)
  post :
    self.manufacturer.lastModified = Now()
```

DeleteManufacturer



```
context  DeleteManufacturer::effect()
  post deleteManufacturer:
    not self.manufacturer@pre.oclIsKindOf(OclAny)
  post changeProductsToOutOfStock:
    deleteProds implies
      manufacturer@pre.product@pre ->
        forAll(status = ProductStatus::outOfStock)
```

Example test program

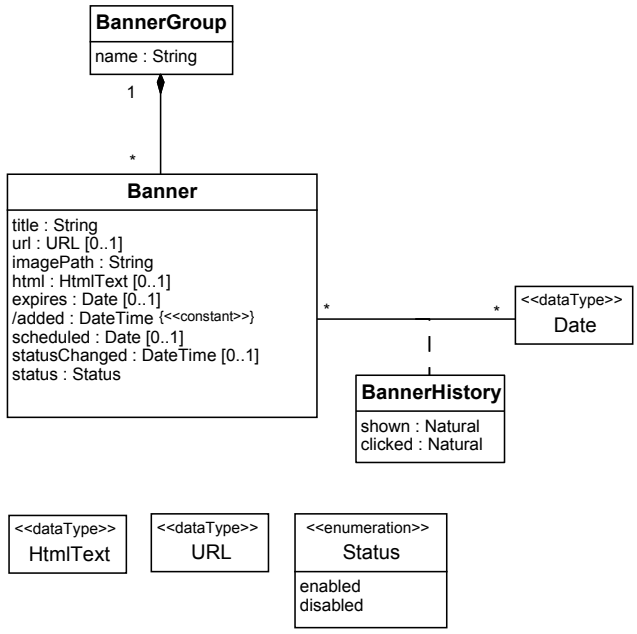
```
testprogram ManufacturersManagement{

        //Test cases are based on a multilingual online shop with two languages
        spanish := new Language(name:='Spanish', code:='ES');
        english := new Language(name:='English', code:='EN');

        test NewManufacturerWithoutURLs{
                new NewManufacturer(name:='BooksEditorial') may not occur;
        }

        test NewManufacturer{
                //We test a valid invocation of the event
                englishURL:=new URL(url:='bookseditorial.com/english');
                spanishURL:=new URL(url:='bookseditorial.com/spanish');
                nm:=new NewManufacturer(name:='bookseditorial');
                new HasURL(url:=englishURL,languageOfURL:=english,manufacturerURLEvent:=this);
                new HasURL(url:=spanishURL,languageOfURL:=spanish,manufacturerURLEvent:=this);
                nm occurs;
                createdManufacturer := Manufacturer.allInstances->any(name='bookseditorial');
                assert equals createdManufacturer.manufacturerInLanguage
                        ->any(language=english).url.url
                       'bookseditorial.com/english';
                assert equals createdManufacturer.manufacturerInLanguage
                        ->any(language=spanish).url.url
                       'bookseditorial.com/spanish';

                //We cannot create the same manufacturer again
                nm2:=new NewManufacturer(name:='bookseditorial');
                new HasURL(url:=englishURL,languageOfURL:=english,manufacturerURLEvent:=this);
                new HasURL(url:=spanishURL,languageOfURL:=spanish,manufacturerURLEvent:=this);
                nm2 may not occur;
        }

        test EditManufacturer{
                //IB state with already existing manufacturers
                englishURL1:=new URL(url:='bookseditorial.com/english');
                spanishURL1:=new URL(url:='bookseditorial.com/english');
                bookseditorial:=new Manufacturer(name:='bookseditorial');
                miEnglish:=new ManufacturerInLanguage
                                (manufacturer:=bookseditorial,language:=english);
                miEnglish.url:=englishURL1;
                miSpanish:=new ManufacturerInLanguage
                                (manufacturer:=bookseditorial,language:=spanish);
                miSpanish.url:=spanishURL1;

                //We create the manufacturer to be modified
                englishURL2:=new URL(url:='www.salamandra.info');
                spanishURL2:=new URL(url:='www.salamandra.info');
                nm:=new NewManufacturer(name:='Salamandra');
                new HasURL
                    (url:=englishURL2,languageOfURL:=english,manufacturerURLEvent:=this);
                new HasURL
                    (url:=spanishURL2,languageOfURL:=spanish,manufacturerURLEvent:=this);
                nm occurs;
                salamandra:=Manufacturer.allInstances->any(name='Salamandra');
```

```
                assert equals salamandra.name 'Salamandra';
                em:=new EditManufacturer(manufacturer:=salamandra,
                                         name:='Ediciones Salamandra');
                new HasURL(url:=englishURL2,
                           languageOfURL:=english,manufacturerURLEvent:=this);
                new HasURL(url:=spanishURL2,
                           languageOfURL:=spanish,manufacturerURLEvent:=this);
                em occurs;
                assert equals salamandra.name 'Ediciones Salamandra';
                em2:=new EditManufacturer(manufacturer:=salamandra,name:='bookseditorial');
                new HasURL(url:=englishURL2,
                          languageOfURL:=english,manufacturerURLEvent:=this);
                new HasURL(url:=spanishURL2,
                          languageOfURL:=spanish,manufacturerURLEvent:=this));
                em2 may not occur;
        }

        test DeleteManufacturerWithNoProducts{
                englishURL1:=new URL(url:='bookseditorial.com/english');
                spanishURL1:=new URL(url:='bookseditorial.com/english');
                nm:=new NewManufacturer(name:='bookseditorial');
                new HasURL(url:=englishURL1,languageOfURL:=english,
                           manufacturerURLEvent:=this);
                new HasURL(url:=spanishURL1,languageOfURL:=spanish,
                           manufacturerURLEvent:=this);
                nm occurs;
                bookseditorial:=Manufacturer.allInstances->any(name='bookseditorial');
                new DeleteManufacturer(manufacturer:=bookseditorial, deleteProds:=false)
                occurs;
                assert true Manufacturer.allInstances->excludes(bookseditorial);
        }

        abstract test DeleteManufacturerWithProducts(Boolean deleteProds){
                englishURL2:=new URL(url:='www.salamandra.info');
                spanishURL2:=new URL(url:='www.salamandra.info');
                nm:=new NewManufacturer(name:='Salamandra');
                new HasURL(url:=englishURL2,languageOfURL:=english,
                           manufacturerURLEvent:=this);
                new HasURL(url:=spanishURL2,languageOfURL:=spanish,
                           manufacturerURLEvent:=this);
                nm occurs;
                salamandra:=Manufacturer.allInstances->any(name='Salamandra');
                bookNameInEnglish:=new StringDT(string:='The Boy in the Striped Pyjamas');
                bookNameInSpanish:=new StringDT(string:='El niño con el pijama de rayas');
                np:=new NewProduct(manufacturer:=salamandra,netPrice:=30,quantityOnHand:=50);
                new HasNewProductName(nameOfProduct:=bookNameInEnglish,
                                      languageOfProduct:=english,productNameEvent:=this);
                new HasNewProductName(nameOfProduct:=bookNameInSpanish,
                                    languageOfProduct:=spanish,productNameEvent:=this);
                np occurs;
                book:=Product.allInstances->any(productInLanguage
                                  ->exists(name='El niño con el pijama de rayas'));
                new DeleteManufacturer(manufacturer:=salamandra, deleteProds:=$deleteProds)
                occurs;

                assert true Manufacturer.allInstances->excludes(salamandra);

                if $deleteProds
                then assert equals book.status #outOfStock;
                endif
        }

        test DeleteManufacturerWithProducts($deleteProds:=false);
        test DeleteManufacturerWithProducts($deleteProds:=true);
}
```

9.14. Banners

Structural schema

osCommerce allows administrating banners published in the *online* store.



[DR1] *Banner::**added*** is the *DateTime* when the banner was created.

context Banner::added():DateTime
body : Now()

[IC1] A Banner is identified by its title.

context Banner::titleIsUnique: Boolean
body : Banner.allInstances() -> isUnique(title)

[IC2] A Banner Group is identified by its name.

context BannerGroup::nameIsUnique: Boolean
body : BannerGroup.allInstances() -> isUnique(name)

Use Cases

Add a banner

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new banner.

Main Success Scenario:

1. The store administrator provides the details of the new banner:

   [→*NewBanner*]
2. The system validates that the data is correct.
3. The system saves the new banner.

Edit a banner

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a banner.

Main Success Scenario:

1. The store administrator selects the banner to be edited.
2. The store administrator provides the new details of the selected banner:

   [→*EditBanner*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a banner

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a banner.

Main Success Scenario:

1. The store administrator selects the banner to be deleted.
2. The store administrator confirms that he wants to delete the banner:

   [→*DeleteBanner*]
3. The system deletes the banner.

Add a banner group

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new banner group.

Main Success Scenario:

1. The store administrator provides the details of the new banner group:

   [→*NewBannerGroup*]
2. The system validates that the data is correct.
3. The system saves the new banner.

Edit a banner group

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a banner group.

Main Success Scenario:

1. The store administrator selects the banner group to be edited.
2. The store administrator provides the new details of the selected banner group:

   [→*EditBannerGroup*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a banner group

Primary Actor: Store administrator

Precondition: The banner group doesn't contain any banners.

Trigger: The store administrator wants to delete a banner.

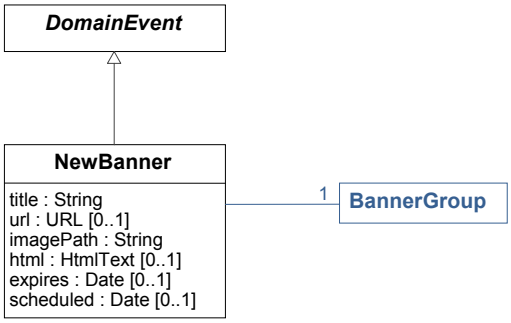
Main Success Scenario:

1. The store administrator selects the banner group to be deleted.
2. The store administrator confirms that he wants to delete the banner group:

   [→*DeleteBannerGroup*]
3. The system deletes the banner.

Events

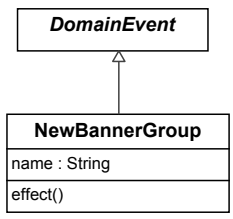
NewBanner



«IniIC»
context NewBanner::bannerDoesNotExist(): Boolean
body : **not** Banner.allInstances() ->exists (b | b.title= self.title)

context NewBanner::effect()
post :
b.oclIsNew() **and**
b.oclIsTypeOf(Banner) **and**
b.title = self.title **and**
b.url = self.url **and**
b.imagePath = self.imagePath **and**
b.html = self.html **and**
b.expires = self.expires **and**
b.scheduled = self.scheduled **and**
b.status = BannerStatus::enabled **and**
b.bannerGroup=self.bannerGroup

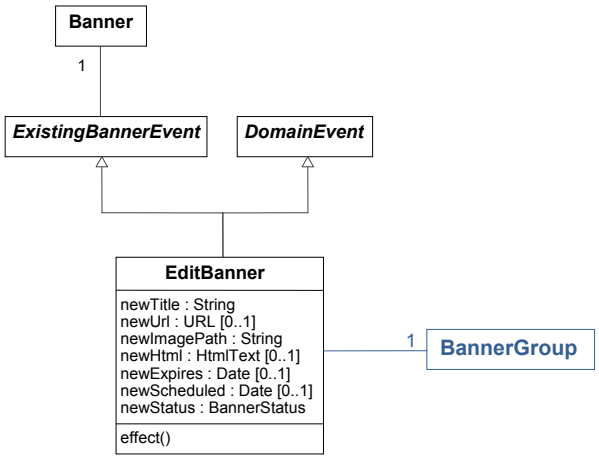
NewBannerGroup



«IniIC»
context NewBannerGroup::bannerGroupDoesNotExist(): Boolean
body : not BannerGroup.allInstances() ->exists (bg | bg.name= self.name)

context NewBannerGroup::effect()
post :
bg.oclIsNew() **and**
bg.oclIsTypeOf(BannerGroup) **and**
bg.name = self.name

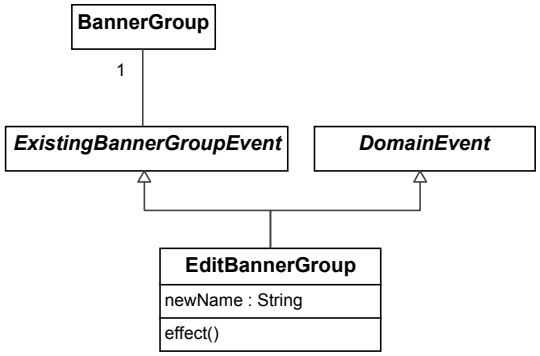
EditBanner



«IniIC»
context EditBanner::bannerDoesNotExist():Boolean
body: (Banner.allInstances - Set{self.banner}).title->excludes(self.newTitle)

context EditBanner::effect()
post :
self.banner.title = self.newTitle **and**
self.banner.url = self.newUrl **and**
self.banner.imagePath = self.newImagePath **and**
self.banner.html = self.newHtml **and**
self.banner.expires = self.newExpires **and**
self.banner.scheduled = self.newScheduled **and**
self.banner.status = self.newStatus **and**
self.banner.bannerGroup=self.bannerGroup
post :
self.banner@pre.status <> self.newStatus **implies** self.banner.statusChanged = Now()

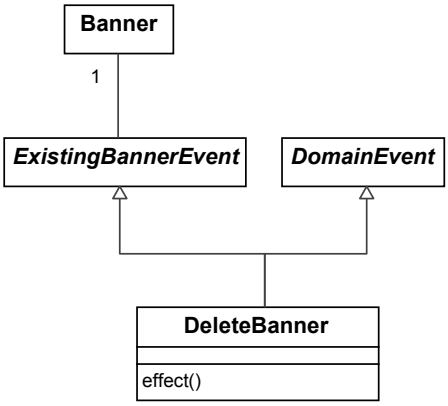
EditBannerGroup



«IniIC»
context EditBannerGroup::bannerGroupDoesNotExist():Boolean
  body: (BannerGroup.allInstances - Set{self.bannerGroup}).name->excludes(self.newName)

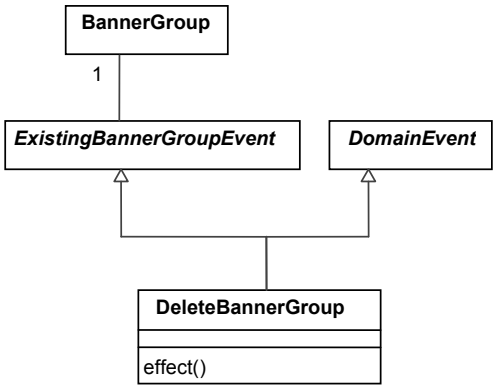
context EditBannerGroup::effect()
  post : self.bannerGroup.name = self.newName

DeleteBanner



context DeleteBanner::effect()
  post : not self.banner@pre.oclIsKindOf(OclAny)

DeleteBannerGroup



«IniIC»
context DeleteBannerGroup::BannerGroupIsEmpty():Boolean
body : self.bannerGroup.banner -> isEmpty()

context DeleteBannerGroup::effect()
post : **not** self.bannerGroup@pre.oclIsKindOf(OclAny)

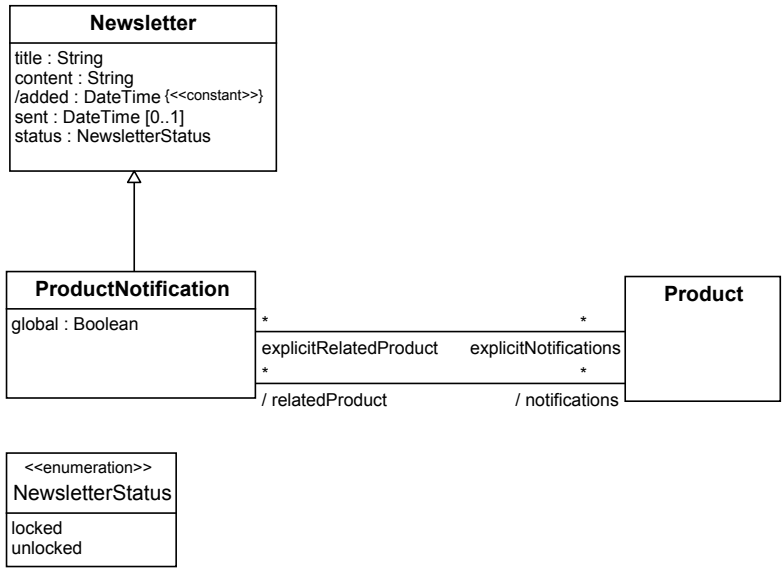
Example test program

```
testprogram BannersManagement{

        test NewBannerGroup{
                new NewBannerGroup(name:='Advertisements') occurs;
                //We cannot create an already existing banner group
                new NewBannerGroup(name:='Advertisements') may not occur;
        }

        test EditBannerGroup{
                new NewBannerGroup(name:='Advertisements') occurs;
                bgroup:=BannerGroup.allInstances->any(name='Advertisements');
                new EditBannerGroup(bannerGroup:=bgroup,newName:='TopAdvertisements') occurs;
                assert equals bgroup.name 'TopAdvertisements';

                //We can edit a banner group without changes
                new EditBannerGroup(bannerGroup:=bg,newName:='TopAdvertisements') occurs;

                //We cannot create duplicates when editing a banner group
                new NewBannerGroup(name:='ChristmasSpecials') occurs;
                new EditBannerGroup(bannerGroup:=bgroup,newName:='ChristmasSpecials')
               may not occur;
        }

        test BannerGroupRequiredForEachBanner{
                new Banner(title:='ChristmasSpecialOffer', imagePath:='special.jpg');
                check inconsistency;
        }

        test NewBanner{
                bg:=new BannerGroup(name:='Advertisements');
                new NewBanner(title:='ChristmasSpecialGift',bannerGroup:=bg) occurs;
                //We cannot create already existing banners
                new NewBanner(title:='ChristmasSpecialGift',bannerGroup:=bg) may not occur;
        }
```

```
        test EditBanner{
                bg:=new BannerGroup(name:='Advertisements');
                bg2:=new BannerGroup(name:='CustomerFidelityCampaign');
                b1:=new Banner(title:='WinTheSpecialPrix', bannerGroup:=bg);
                new EditBanner(banner:=b1,newTitle:='WinACar!', newBannerGroup:=bg2) occurs;
                assert equals b1.title 'WinACar!';
                assert equals b1.bannerGroup bg2;

                //We cannot generate duplicate banners when edit
                b2:=new Banner(title:='25% off', bannerGroup:=bg2);
                new EditBanner(banner:=b2,newTitle:='25% off', newBannerGroup:=bg2) occurs;
                new EditBanner(banner:=b1,newTitle:='25% off', newBannerGroup:=bg)
                may not occur;
        }

        test deleteBanner{
                bg:=new BannerGroup(name:='Advertisements');
                b1:=new Banner(title:='NewBabiesSection', bannerGroup:=bg);
                new DeleteBanner(banner:=b1) occurs;
                assert true Banner.allInstances->size()=0;
        }

        test deleteBannerGroup{
                //A banner group with banners cannot be deleted
                bg:=new BannerGroup(name:='Sponsors');
                b1:=new Banner(title:='ParisTourism', bannerGroup:=bg);
                new DeleteBannerGroup(bannerGroup:=bg) may not occur;
                new DeleteBanner(banner:=b1) occurs;
                new DeleteBannerGroup(bannerGroup:=bg) occurs;
        }
}
```

9.15. Newsletters

osCommerce allows store administrators sending emails and product notifications to customers.



[DR1] *ProductNotification::**notifications*** is the set of implied products in the notification.

```
context ProductNotification::notifications():Set(Product)
body :
   if self.global then Product.allInstances()
   else self.explicitNotifications
   endif
```

[DR2] *ProductNotification::**added*** is the *DateTime* when the newsletter was created.

```
context Newsletter::added():DateTime
 body : Now()
```

[IC1] A Newsletter is identified by its title.

```
context Newsletter::titleIsUnique: Boolean
 body :  Newsletter.allInstances() -> isUnique(title)
```

Use Cases

Create a newsletter

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to create a new newsletter.

Main Success Scenario:

1. The store administrator selects the type of the newsletter (newsletter or product nofitification).
2. The store administrator provides the title and the content of the newsletter:

   [→*NewNewsletter*]

   [→*NewProductNotification*]
3. The system validates that the data is correct.
4. The system saves the newsletter.

Edit a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is unlocked.

Trigger: The store administrator wants to edit a newsletter.

Main Success Scenario:

1. The store administrator selects the newsletter to be edited.
2. The store administrator provides the new details of the selected newsletter:

   [→*EditNewsletter*]

   [→*EditProductNotification*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is unlocked.

Trigger: The store administrator wants to delete a newsletter.

Main Success Scenario:

1. The store administrator selects the newsletter to be deleted.
2. The store administrator confirms that he wants to delete the newsletter:

   [→*DeleteNewsletter*]
3. The system deletes the newsletter.

Lock a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is unlocked.

Trigger: The store administrator wants to indicate to the other administrators that a newsletter is pending to be delivered.

Main Success Scenario:

1. The store administrator selects the newsletter to be locked.

   [→*LockNewsletter*]
2. The system saves the change.

Unlock a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is locked.

Trigger: The store administrator wants to indicate to the other administrators that a newsletter ceases to be locked.

Main Success Scenario:

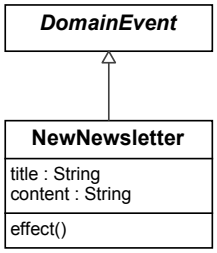
1. The store administrator selects the newsletter to be unlocked.

   [→*UnlockNewsletter*]

2. The system saves the change.

Events

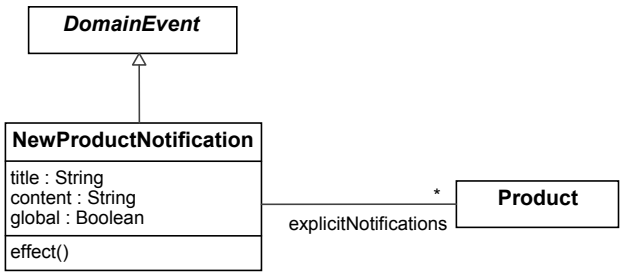
NewNewsletter



«IniIC»
context NewNewsletter::newsletterDoesNotExist(): Boolean
body : **not** Newsletter.allInstances() -> exists (n | n.title=self.title)

context NewNewsletter::effect()
post :
n.oclIsNew() **and**
n.oclIsTypeOf(Newsletter) **and**
n.title = self.title **and**
n.content = self.content **and**
n.status = NewsletterStatus::unlocked

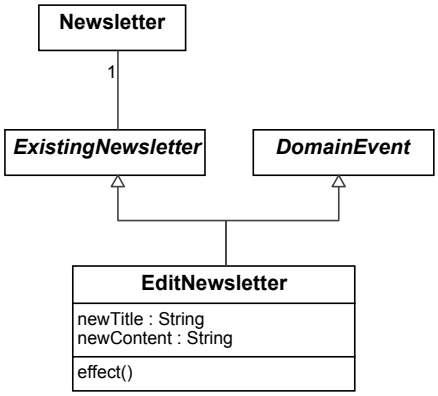
NewProductNotification



«IniIC»
context NewProductNotification::ProductNotificationDoesNotExist(): Boolean
body : not Newsletter.allInstances() -> exists (n | n.title = self.title)

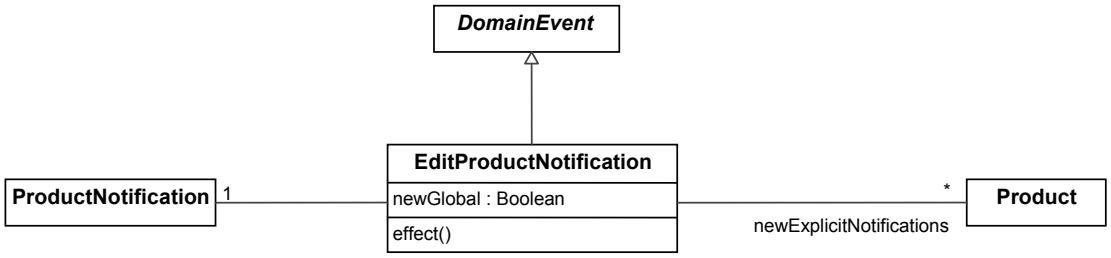
```
context  NewProductNotification::effect()
  post :
     n.oclIsNew()  and
     n.oclIsTypeOf(ProductNotification) and
     n.title = self.title and
     n.content = self.content and
     n.global = self.global and
     n.explicitNotifications = self.explicitNotifications and
     n.status = self.NewsletterStatus::unlocked
```

EditNewsletter



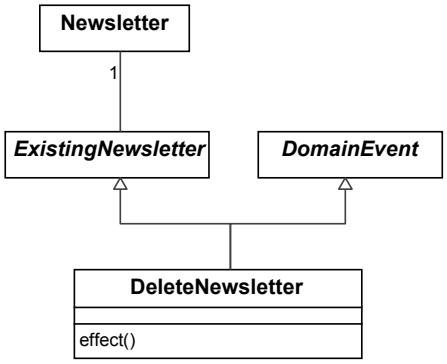

```
«IniIC»
context  EditNewsletter::newsletterIsUnlocked():Boolean
   body: self.newsletter.status = Status::unlocked

«IniIC»
context EditNewsletter::newsletterDoesNotExist():Boolean
   body:  (Newsletter.allInstances - Set{self.newsletter}).title->excludes(self.newTitle)

context  EditNewsletter::effect()
  post :
     newsletter.title = self.newTitle and
     newsletter.content = self.newContent
```

EditProductNotification



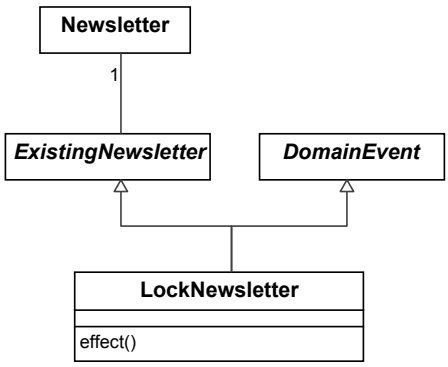
```
context EditProductNotification::effect()
  post :
     self.productNotification.global = self.newGlobal and
     self.productNotification.explicitNotifications = self.newExplicitNotifications
```

DeleteNewsletter



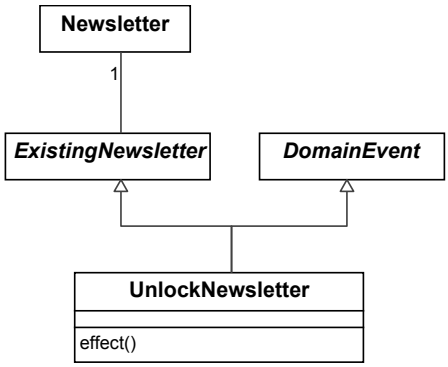

```
«IniIC»
context DeleteNewsletter::newsletterIsUnlocked():Boolean
   body: self.newsletter.status = Status::unlocked

context DeleteNewsletter::effect()
  post : not self.newsletter@pre.oclIsKindOf(OclAny)
```

LockNewsletter




```
«IniIC»
context LockNewsletter::newsletterIsNotLocked():Boolean
   body: self.newsletter.status <> Status::locked

context LockNewsletter::effect()
  post : self.newsletter.status = NewsletterStatus::locked
```

UnlockNewsletter



«IniIC»
context UnlockNewsletter::newsletterIstLocked():Boolean
body: self.newsletter.status <>Status::unlocked

context UnlockNewsletter::effect()
post : self.newsletter.status = NewsletterStatus::unlocked

Example test programs

```
testprogram NewslettersManagement{

        test NewNewsletter{
                new NewNewsletter(title:='NewSection',
                                  content:='Our new sports section is now opened !') occurs;

                //We cannot create an already existing newsletter
                new NewNewsletter(title:='NewSection',
                                   content:='Our new sports section is now opened !')
                may not occur;

                //...even if it is a product notification (because a product noficitaion is
                //is also a newsletter
                p:=new Product;
                new NewProductNotification(title:='NewSection',
                    content:='New section of products similar to p is now opened',
                    explicitNotifications:=p) may not occur;
        }

        test EditNewsletter{
                new NewNewsletter(title:='NewSection',
                                   content:='Our new sports section is now opened !') occurs;
                n1:=Newsletter.allInstances->any(title='NewSection');

               //We cannot lock already locked newsletters
                new LockNewsletter(newsletter:=n1) occurs;
                new LockNewsletter(newsletter:=n1) may not occur;

                //We cannot edit locked newsletters
                new EditNewsletter(newsletter:=n1,newTitle:='NewTitle') may not occur;
                new UnlockNewsletter(newsletter:=n1) occurs;
                new UnlockNewsletter(newsletter:=n1) may not occur;

                //Valid newsletter editions
                new EditNewsletter(newsletter:=n1,newTitle:='NewSection') occurs;
```

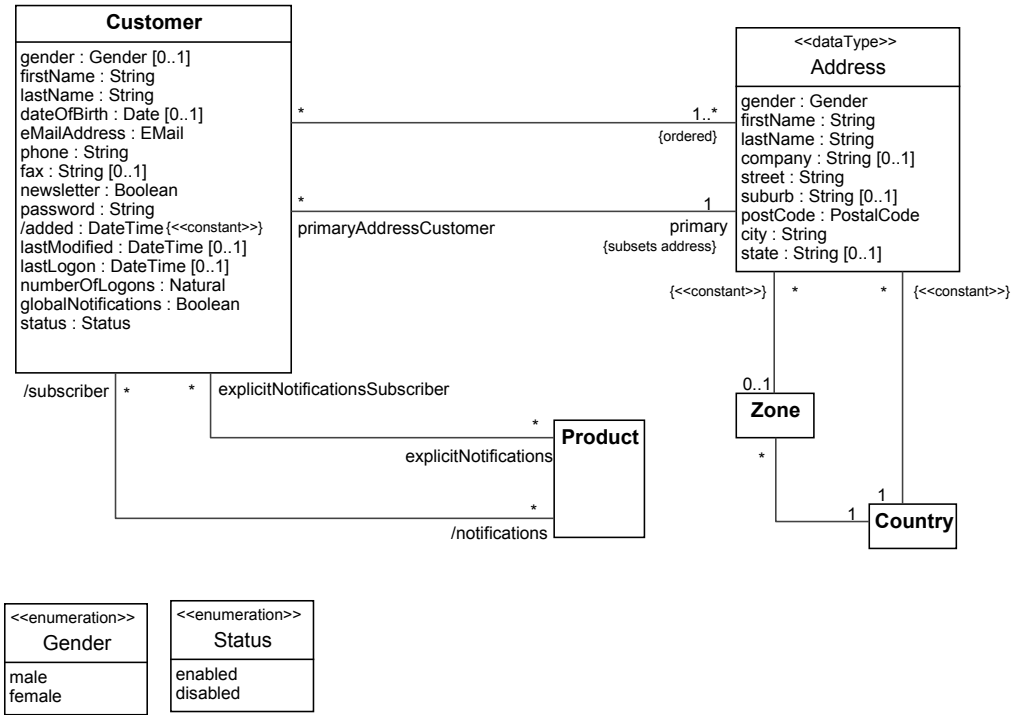
```
                new EditNewsletter(newsletter:=n1,newTitle:='NewSectionAnnouncement') occurs;
                assert equals n.title 'NewSectionAnnouncement';

                //We cannot create duplicates when editing a newsletter
                new NewNewsletter(title:='NewSpringFashionSection',
                       content:='Our new spring fashion section is now opened !') occurs;
                n2:=Newsletter.allInstances->any(title='NewSpringFashionSection');
                new EditNewsletter(newsletter:=n2,newTitle:='NewSectionAnnouncement')
                may not occur;
                }

        test DeleteNewsletter{
                new NewNewsletter(title:='NewSection',
                                  content:='Our new sports section is now opened !') occurs;
                n:=Newsletter.allInstances->any(title='NewSection');

                //A locked newsletter cannot be deleted
                new LockNewsletter(newsletter:=n) occurs;
                new DeleteNewsletter(newsletter:=n) may not occur;

                //Only unlocked newsletter can be deleted
                new UnlockNewsletter(newsletter:=n) occurs;
                new DeleteNewsletter(newsletter:=n) occurs;
                assert true Newsletter.allInstances->excludes(n);
        }
}
```

```
testprogram ProductNotifications{

        //In this test program we exercise the specific properties of product notifications

        aucaSenyorEsteveBook := new Product;
        tirantLoBlancBook := new Product;

        new NewProductNotification(title:='Frankfurt 2007',
                                   content:='Catalan culture will be the guest of honour at
                                   the 2007 Frankfurt Book Fair.',
                                   global:=false,
                                   explicitNotifications := aucaSenyorEsteveBook) occurs;

        pn1:=ProductNotification.allInstances->any(title='Frankfurt 2007');

        test globalNotificationsDisabled{
                //We test the derived relationship notifications using materialitzation
                pn1._notifications:=Set{aucaSenyorEsteveBook};
                check consistency;
        }

        test globalNotificationsEnabled{
                pn1.global:=true;
                //We test the derived relationship notifications using materialitzation
                pn1._notifications:=Set{aucaSenyorEsteveBook,tirantLoBlancBook};
                check consistency;
        }
}
```

9.16. Customers

Structural schema

osCommerce keeps information about customers and their addresses, one of which is the primary address.



[DR1] *Customer::**notifications*** is the set of subscriptions to product notifications.

```
context Customer::notifications():Set(Product)
body :
   if self.globalNotifications then Product.allInstances()
   else self.explicitNotifications
   endif
```

[DR2] *Customer::**added*** is the *DateTime* of the customer creation.

```
context Customer::added():DateTime
  body : Now()
```

[IC1] Customers are identified by their email address.

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

[IC2] Addresses have zone if needed.

```
context Country::addressesHaveZoneIfNeeded(): Boolean
body :  self.zone -> notEmpty() implies self.address -> forAll
        (a | a.state = a.zone.name and self = a.zone.country)
```

Use Cases

Create a customer

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to open an account in the store.

Main Success Scenario:

1. The customer provides the required customer data:

   [→*NewCustomer*]

2. The system validates the customer data.
3. The system saves the new account.

Change password

Primary Actor: Customer

Precondition: The customer is logged in.

Trigger: A customer wants to change his password.

Main Success Scenario:

1. The customer provides the old password.
2. The customer provides the new password twice.

   [→*PasswordChange*]

3. The system validates that the data is correct.
4. The system saves the changes.

Change customer details

Primary Actor: Customer

Precondition: The customer is logged in.

Trigger: A customer wants to change its customer details.

Main Success Scenario:

1. The customer provides the new customer details.
   [→*EditCustomerDetails*]
2. The system validates that the data is correct.
3. The system saves the changes.

Administrate address book

Primary Actor: Customer

Precondition: The customer is logged in and the number of addresses is less than the maximum number of address entries permitted.

Trigger: A customer wants to view or change the address book.

Main Success Scenario:

1. The system displays the current address book entries of the customer.
2. The customer selects an address book entry to be edited :
   [→*EditCustomerAddress*]
3. The system validates that the data is correct.
4. The system saves the changes and displays the new address book.

   The customer repeats steps 1-4 until he is done.

Extensions:

2a. The customer doesn't want to change the address book:
   2a1. The use case ends.

2b. The customer wants to add a new address book entry:
   2b1. The customer provides the required data:
      [→*NewCustomerAddress*]
   2b2. The use case continues at step 3.

2c. The customer wants to delete an address book entry:
   2c1. The customer selects the address book entry:
      [→*DeleteCustomerAddress*]
   2c2. The use case continues at step 3.

2d. The customer wants to change the default address book entry:
   2d1. The customer selects the new default address book entry:

[→*PrimaryCustomerAddressChange*]

2d2. The use case continues at step 3.

Edit a customer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a customer.

Main Success Scenario:

1. The store administrator selects the customer to be edited.
2. The store administrator provides the new details of the selected customer:

   [→*EditCustomer*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a customer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a customer.

Main Success Scenario:

1. The store administrator selects the customer to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the customer:

   [→*DeleteCustomer*]
4. The system deletes the customer and their addresses, reviews, notification subscriptions and shopping carts.

Extensions:

3a. The customer has orders:

3a1. The system changes the status of the customer to disable.

[→*CustomerStatusChange*]

3a2. The system deletes customer's addresses, reviews, notification subscriptions and shopping carts.

3a3. The use case ends.

Administrate subscriptions

Primary Actor: Customer

Precondition: The customer is logged in.

Trigger: A customer wants to view or change their product notification subscriptions.

Main Success Scenario:

1. The system displays the details of the current product notification subscriptions of the customer.
2. The customer adds a new product subscription:

   [→*NewProductNotificationSubscription*]

3. The system validates that the data is correct.
4. The system saves the changes and displays the new product notification subscriptions.

   The customer repeats steps 1-4 until he is done.

Extensions:

2a. The customer doesn't want to change their product notification subscriptions:

   2a1. The use case ends.

2b. The customer wants to be subscribed or unsubscribed to all product notifications:

   [→ *EditGlobalNotifications*]

2c. The customer wants to delete a product notification subscription:

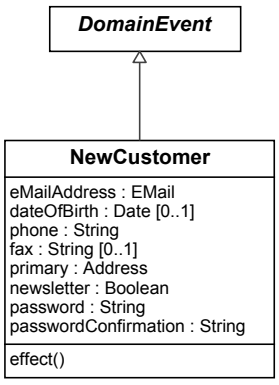
   2c1. The customer selects the product:

      [→*DeleteProductNotificationSubscription*]

   2c2. The use case continues at step 3.

Events

NewCustomer

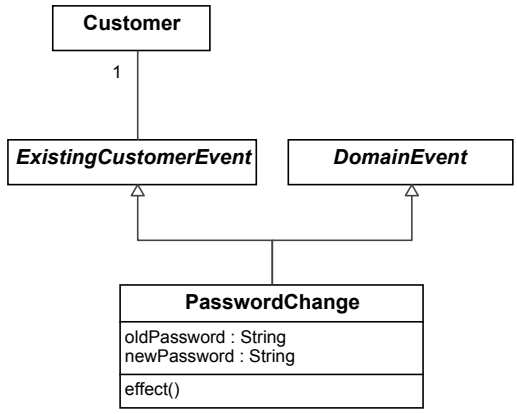


```
«IniIC»
context NewCustomer::customerDoesNotExist(): Boolean
  body : not Customer.allInstances() -> exists (c | c.eMailAddress = self.eMailAddress)

«IniIC»
context NewCustomer::passwordCorrect(): Boolean
  body : password = passwordConfirmation

«IniIC»
context NewCustomer::firstNameRight(): Boolean
  body : self.primary.firstName.size() >= MinimumValues.firstName

«IniIC»
context NewCustomer::lastNameRight(): Boolean
  body : self.primary.lastName.size() >= MinimumValues.lastName

«IniIC»
context NewCustomer::dateOfBirthRight(): Boolean
  body :   CustomerDetails.dateOfBirth implies
           self.dateOfBirth -> notEmpty() and
           self.dateOfBirth.size() >= MinimumValues.dateOfBirth

«IniIC»
context NewCustomer::genderRight(): Boolean
  body : CustomerDetails.gender implies self.gender->notEmpty()

«IniIC»
context NewCustomer::suburbRight(): Boolean
  body : CustomerDetails.suburb implies self.suburb->notEmpty()

«IniIC»
context NewCustomer::eMailRight(): Boolean
  body : self.eMailAddress.size() >= MinimumValues.eMailAddress

«IniIC»
context NewCustomer::streetAddressRight(): Boolean
  body : self.primary.street.size() >= MinimumValues.streetAddress
«IniIC»
context NewCustomer::companyRight(): Boolean
  body :
      CustomerDetails.company implies
      self.primary.company -> notEmpty() and
      self.primary.company.size() >= MinimumValues.companyName

«IniIC»
context NewCustomer::postCodeRight(): Boolean
  body : self.primary.postCode.size() >= MinimumValues.postCode

«IniIC»
context NewCustomer::cityRight(): Boolean
  body : self.primary.city.size() >= MinimumValues.city

«IniIC»
context NewCustomer::stateRight(): Boolean
  body :
      CustomerDetails.state implies
      self.primary.state -> notEmpty() and
      self.primary.state.size() >= MinimumValues.state
```

«*IniIC*»
context NewCustomer::telephoneRight(): Boolean
body : self.telephone.size() >= MinimumValues.telephoneNumber

«*IniIC*»
context NewCustomer::passwordRight(): Boolean
body : self.password.size() >= MinimumValues.password
context NewCustomer::effect()
post :
c.oclIsNew() **and**
c.oclIsTypeOf(Customer) **and**
c.gender = self.primary.gender **and**
c.firstName = self.primary.firstName **and**
c.lastName = self.primary.lastName **and**
c.dateOfBirth = self.dateOfBirth **and**
c.eMailAddress = self.eMailAddress **and**
c.phone = self.phone **and**
c.fax = self.fax **and**
c.newsletter = self.newsletter **and**
c.password = self.password **and**
c.numberOfLogons = 0 **and**
c.address = Set{primary} **and**
c.primary = primary

PasswordChange

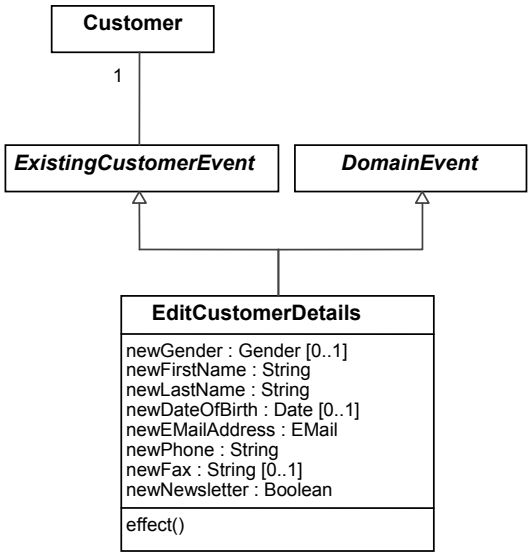


«*IniIC*»
context ChangePassword::passwordRight(): Boolean
body : self.password.size() >= MinimumValues.password

«*IniIC*»
context ChangePassword::OldPasswordIsCorrect(): Boolean
body : customer.password = self.oldPassword

context ChangePassword::effect()
post : self.customer.password = self.newPassword

EditCustomerDetails



«*IniIC*»
context EditCustomerDetails::firstNameRight(): Boolean
body : self.newFirstName.size() >= MinimumValues.firstName

«*IniIC*»
context EditCustomerDetails::lastNameRight(): Boolean
body : self.newLastName.size() >= MinimumValues.lastName

«*IniIC*»
context EditCustomerDetails::dateOfBirthRight(): Boolean
body :
CustomerDetails.dateOfBirth **implies**
self.newDateOfBirth->notEmpty()
self.newDateOfBirth.size() >= MinimumValues.dateOfBirth

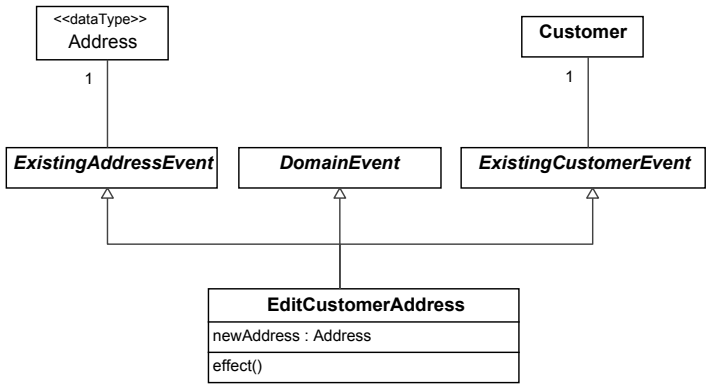
«*IniIC*»
context EditCustomerDetails::genderRight(): Boolean
body : CustomerDetails.gender **implies** self.newGender->notEmpty()

«*IniIC*»
context EditCustomerDetails::eMailRight(): Boolean
body : self.newEMailAddress.size() >= MinimumValues.eMailAddress

«*IniIC*»
context EditCustomerDetails::telephoneRight(): Boolean
body : self.newTelephone.size() >= MinimumValues.telephoneNumber

context EditCustomerDetails::effect()
post :
customer.gender = self.newGender **and**
customer.firstName = self.newFirstName **and**
customer.lastName = self.newLastName **and**
customer.dateOfBirth = self.newDateOfBirth **and**
customer.eMailAddress = self.newEMailAddress **and**
customer.phone = self.newPhone **and**
customer.fax = self.newFax **and**
customer.newsletter = self.newNewsletter

EditCustomerAddress



«IniIC»
context EditCustomerAddress::AddressOfCustomer(): Boolean
body : self.customer.address -> includes(self.address)

«IniIC»
context EditCustomerAddress::firstNameRight(): Boolean
body : self.newAdress.firstName.size() >= MinimumValues.firstName

«IniIC»
context EditCustomerAddress::lastNameRight(): Boolean
body : self. newAdress.lastName.size() >= MinimumValues.lastName

«IniIC»
context EditCustomerAddress::genderRight(): Boolean
body : CustomerDetails.gender **implies** self. newAdress.gender->notEmpty()

«IniIC»
context EditCustomerAddress::suburbRight(): Boolean
body : CustomerDetails.suburb **implies** self. newAdress.suburb->notEmpty()

«IniIC»
context EditCustomerAddress::streetAddressRight(): Boolean
body : self.newAdress.street.size() >= MinimumValues.streetAddress

«IniIC»
context EditCustomerAddress::companyRight(): Boolean
body :
CustomerDetails.company **implies**
self.newAdress.company -> notEmpty() **and**
self.newAdress.company.size() >= MinimumValues.companyName

«IniIC»
context EditCustomerAddress::postCodeRight(): Boolean
body : self.newAdress.postCode.size() >= MinimumValues.postCode

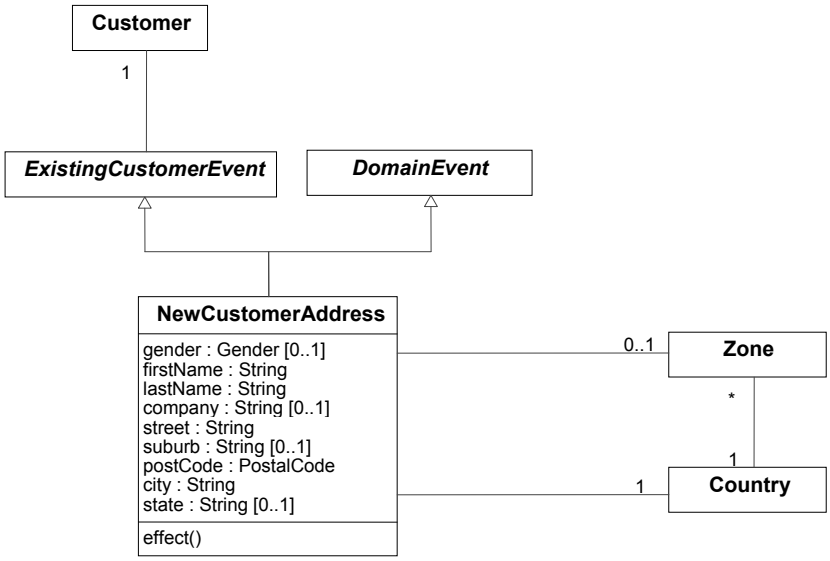
«IniIC»
context EditCustomerAddress::cityRight(): Boolean
body : self.newAdress.city.size() >= MinimumValues.city

«*IniIC*»
context EditCustomerAddress::stateRight(): Boolean
body :
CustomerDetails.state **implies**
self.newAdress.state -> notEmpty() **and**
self.newAdress.state.size() >= MinimumValues.state

«*IniIC*»
context EditCustomerAddress::addressesHaveZoneIfNeeded(): Boolean
body :
self.newAdress.zone -> notEmpty() **implies**
self.newAdress.state = self.newAdress.zone.name **and**
self.newAdress.country = self.newAdress.zone.country

context EditCustomerAddress::effect()
post :
self.customer.address -> excludes(self.address) **and**
self.customer.address ->includes(self.newAddress)

NewCustomerAddress



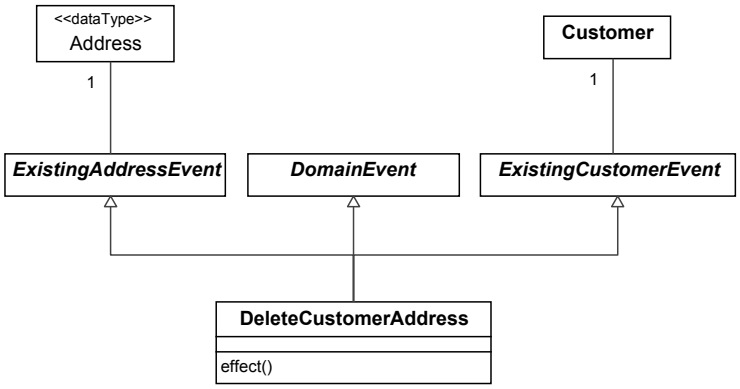
«*IniIC*»
context NewCustomerAddress::firstNameRight(): Boolean
body : self.primary.firstName.size() >= MinimumValues.firstName

«*IniIC*»
context NewCustomerAddress::lastNameRight(): Boolean
body : self.primary.lastName.size() >= MinimumValues.lastName

«*IniIC*»
context NewCustomerAddress::genderRight(): Boolean
body : CustomerDetails.gender **implies** self.gender->notEmpty()

```
«IniIC»
context NewCustomerAddress::suburbRight(): Boolean
  body : CustomerDetails.suburb implies self.suburb->notEmpty()

«IniIC»
context NewCustomerAddress::streetAddressRight(): Boolean
  body : self.primary.street.size() >= MinimumValues.streetAddress

«IniIC»
context NewCustomerAddress::companyRight(): Boolean
  body :
     CustomerDetails.company implies
      self.primary.company -> notEmpty() and
      self.primary.company.size() >= MinimumValues.companyName

«IniIC»
context NewCustomerAddress::postCodeRight(): Boolean
  body : self.primary.postCode.size() >= MinimumValues.postCode

«IniIC»
context NewCustomerAddress::cityRight(): Boolean
  body : self.primary.city.size() >= MinimumValues.city

«IniIC»
context NewCustomerAddress::stateRight(): Boolean
  body :
      CustomerDetails.state implies
      self.primary.state -> notEmpty() and
      self.primary.state.size() >= MinimumValues.state

«IniIC»
context NewCustomerAddress::addressesHaveZoneIfNeeded(): Boolean
  body :
      self.country.zone->size()>0
      implies
     (self.state = self.zone.name and
      self.country = self.zone.country)

«IniIC»
context NewCustomerAddress::numberOfAddressesRight(): Boolean
  body : self.customer.address -> size() < MaximumValues.addressBookEntries


context  NewCustomerAddress::effect()
   post :
         Address.allInstances() ->exists (a |
         a.gender = self.gender and
         a.firstName = self.firstName and
         a.lastName = self.lastName and
         a.company = self.company and
         a.street = self.street and
         a.suburb = self.suburb and
         a.postCode = self.postCode and
         a.city = self.city and
         a.state = self.state and
         a.zone = self.zone and
         a.country = self.country and
         self.customer.address -> includes(a))
```

DeleteCustomerAddress



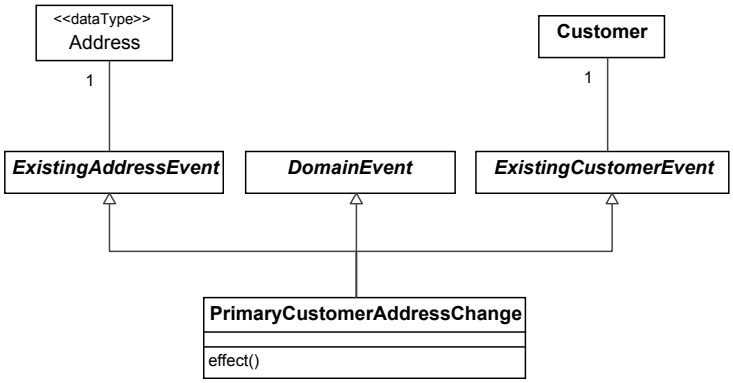
«InilC»
context DeleteCustomerAddress::AddressOfCustomer(): Boolean
body : self.customer.address -> includes(self.address)

«InilC»
context DeleteCustomerAddress::AtLeastTwoAddresses(): Boolean
body : self.customer.address.size() >= 2

«InilC»
context DeleteCustomerAddress::PrimaryAddressCannotBeDeleted():Boolean
self.address <> self.customer.primary

context DeleteCustomerAddress::effect()
post : self.customer.address -> excludes(self.address)

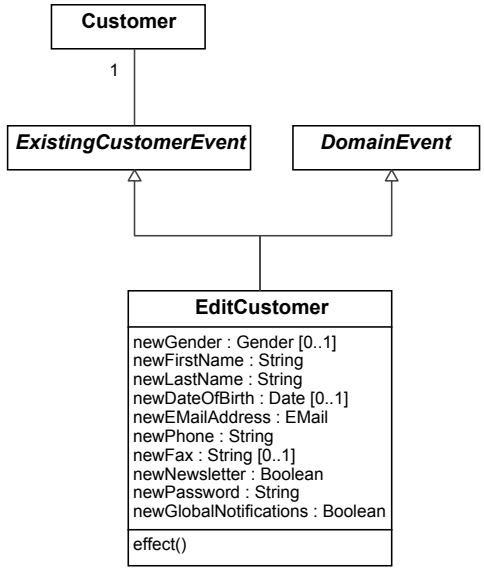
PrimaryCustomerAddressChange



«InilC»
context PrimaryCustomerAddressChange::AddressOfCustomer(): Boolean
body : self.customer.address -> includes(self.address)

context PrimaryCustomerAddressChange::effect()
post : self.customer.primary = self.address

EditCustomer



«IniIC»
context EditCustomer::firstNameRight(): Boolean
body : self.newFirstName.size() >= MinimumValues.firstName

«IniIC»
context EditCustomer::lastNameRight(): Boolean
body : self.newLastName.size() >= MinimumValues.lastName

«IniIC»
context EditCustomer::dateOfBirthRight(): Boolean
body :
CustomerDetails.dateOfBirth **implies**
self.newDateOfBirth->notEmpty() **and**
self.newDateOfBirth.size() >= MinimumValues.dateOfBirth

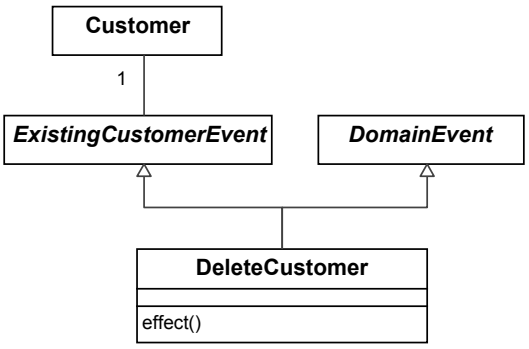
«IniIC»
context EditCustomer::genderRight(): Boolean
body : CustomerDetails.gender **implies** self.newGender->notEmpty()

«IniIC»
context EditCustomer::eMailRight(): Boolean
body : self.newEMailAddress.size() >= MinimumValues.eMailAddress
«IniIC»
context EditCustomer::telephoneRight(): Boolean
body : self.newTelephone.size() >= MinimumValues.telephoneNumber

context EditCustomer::effect()
post :
customer.gender = self.newGender **and**
customer.firstName = self.newFirstName **and**
customer.lastName = self.newLastName **and**
customer.dateOfBirth = self.newDateOfBirth **and**
customer.eMailAddress = self.newEMailAddress **and**

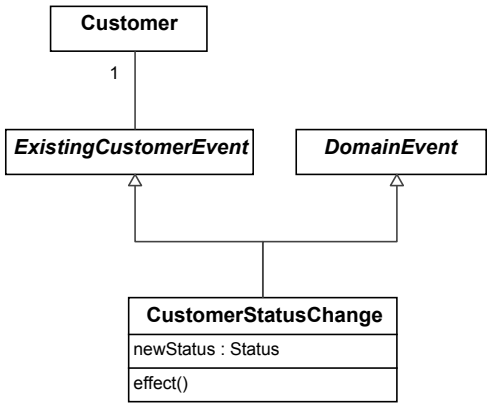
```
    customer.phone = self.newPhone and
    customer.fax = self.newFax and
    customer.newsletter = self.newNewsletter and
    customer.password = self.newPassword and
    customer.globalNotifications = self.newGlobalNotifications and
  post :
    customer.lastModified = Now()
```

DeleteCustomer



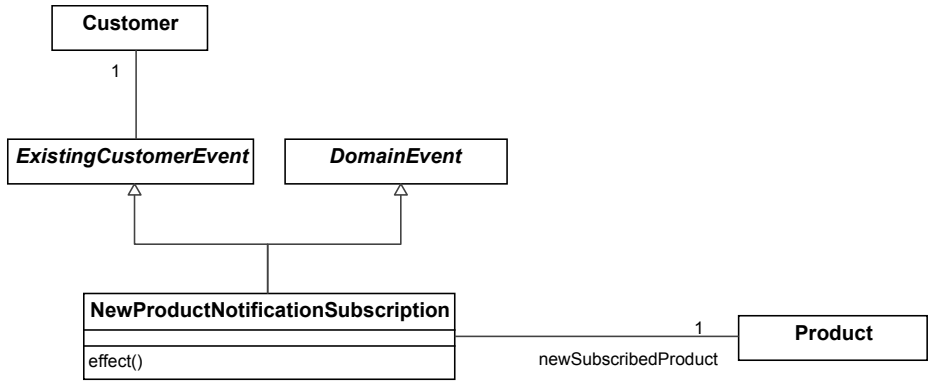

```
context  DeleteCustomer::effect()
  post deleteCustomer:
    not customer@pre.oclIsKindOf(OclAny)
  post deleteReviewsAndShoppingCart:
    not customer@pre.review@pre -> forAll (r | r.oclIsKindOf(OclAny)) and
    (customer@pre.customerShoppingCart->notEmpty()
    implies
    not customer@pre.customerShoppingCart@pre.oclIsKindOf(OclAny))
```

CustomerStatusChange




```
context  CustomerStatusChange::effect()
  post :  self.customer.status = self.newStatus
```

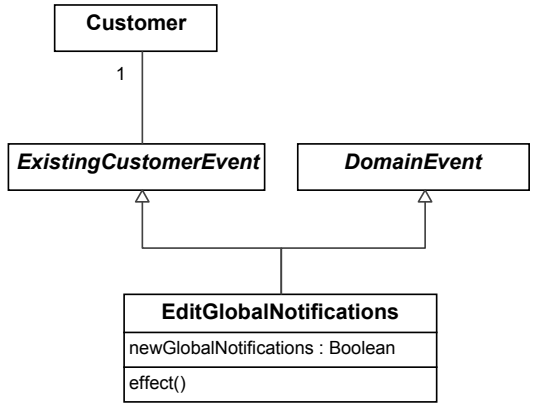
NewProductNotificationSubscription



«InitC»
context NewProductNotificationSubscription::ProductIsUnsubscribed(): Boolean
body :
not self.customer.globalNotifications **and**
self.customer.explicitNotifications -> excludes(self.newSubscribedProduct)

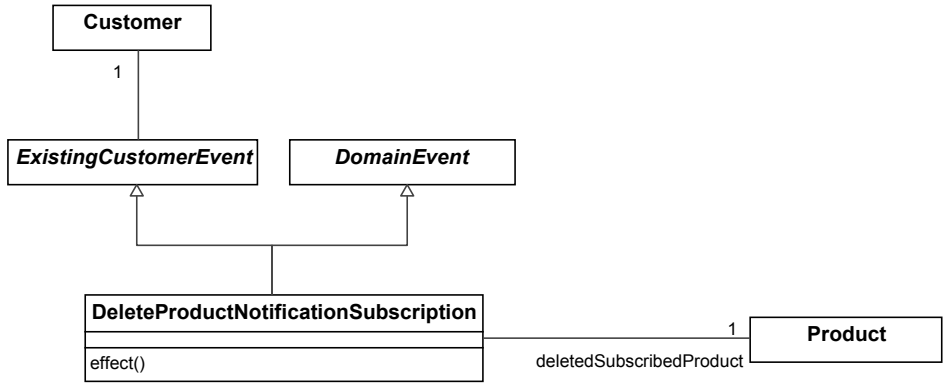
context NewProductNotificationSubscription::effect()
post : self.customer.explicitNotifications -> includes(self.newSubscribedProduct)

EditGlobalNotifications



context EditGlobalNotifications::effect()
post : self.customer.globalNotifications = self.newGlobalNotifications

DeleteProductNotificationSubscription



context DeleteProductNotificationSubscription::effect()
post : customer.explicitNotifications -> excludes(self.deletedSubscribedProduct)

Example test programs

```
testprogram NewCustomer{

        textConfigurationValues := new MinimumValues, MaximumValues;
        textConfigurationValues.firstName:=1;
        textConfigurationValues.lastName:=1;
        textConfigurationValues.dateOfBirth:=6;
        textConfigurationValues.eMailAddress:=1;
        textConfigurationValues.streetAddress:=1;
        textConfigurationValues.companyName:=0;
        textConfigurationValues.postCode:=1;
        textConfigurationValues.city:=1;
        textConfigurationValues.state:=1;
        textConfigurationValues.telephoneNumber:=9;
        textConfigurationValues.password:=4;
        textConfigurationValues.addressBookEntries:=2;

        customerDetailsConfiguration :=  new CustomerDetails;
        customerDetailsConfiguration.gender:=false;
        customerDetailsConfiguration.dateOfBirth:=false;
        customerDetailsConfiguration.company:=true;
        customerDetailsConfiguration.state:=false;
        customerDetailsConfiguration.suburb:=false;
        d:= new Date(date:='X/XX/XXXX');

        abstract test validNewCustomer(String mail, String phone, String company,
                                       String fax, String firstName, String lastName,
                                       String street, String postCode, String city,
                                       String country, Boolean newsletter,
                                       String password, String passwordConfirmation){
                e := new EMail(eMail:=$mail);
                pc:= new PostalCode(postalCode:=$postCode);
                c := new Country(name:=$country);
                a := new Address
                        (firstName:=$firstName, lastName:=$lastName, company:=$company,
                         street:=$street, postCode:=pc, city:=$city, country:=c);
                new NewCustomer(eMailAddress:=e, dateOfBirth:=d, phone:=$phone,
                                fax:=$fax, primary:=a, newsletter:=$newsletter,
                                password:=$password,
                                passwordConfirmation:=$passwordConfirmation) occurs;
        }
```

```
        abstract test invalidNewCustomer(String mail, String phone, String company,
                                         String fax, String firstName, String lastName,
                                         String street, String postCode, String city,
                                         String country, Boolean newsletter, String password,
                                         String passwordConfirmation){
                e := new EMail(eMail:=$mail);
                pc:= new PostalCode(postalCode:=$postCode);
                c := new Country(name:=$country);
                a := new Address
                        (firstName:=$firstName, lastName:=$lastName, company:=$company,
                         street:=$street, postCode:=pc, city:=$city, country:=c);
                new NewCustomer(eMailAddress:=e, dateOfBirth:=d, phone:=$phone,
                                fax:=$fax, primary:=a, newsletter:=$newsletter,
                                password:=$password,
                                passwordConfirmation:=$passwordConfirmation) may not occur;
        }

        //We can easily test the NewCustomer event in different valid or invalid contexts

        test validNewCustomer
            ($mail:='atort@lsi.upc.edu', $phone:='XXXXXXXXX', $company:='UPC',
             $fax:='XXXXXXXXX', $firstName:='Albert', $lastName:='Tort',
             $street:='Jordi Girona,1', $postCode:='08034', $city:='Barcelona',
             $country:='Espanya', $newsletter:=true, $password:='password',
             $passwordConfirmation:='password');

        test validNewCustomer
             ($mail:='olive@lsi.upc.edu', $phone:='XXXXXXXXX', $company:='UPC',
              $fax:='XXXXXXXXX', $firstName:='Antoni', $lastName:='Olive',
              $street:='Jordi Girona,1', $postCode:='08034', $city:='Barcelona',
              $country:='Espanya', $newsletter:=false, $password:='password',
              $passwordConfirmation:='password');

        //Incorrect password confirmation
        test invalidNewCustomer
            ($mail:='olive@lsi.upc.edu', $phone:='XXXXXXXXX', $company:='UPC',
             $fax:='XX XXX XX XX', $firstName:='Antoni', $lastName:='Olive',
             $street:='Jordi Girona,1', $postCode:='08034', $city:='Barcelona',
             $country:='Espanya', $newsletter:=false, $password:='password',
             $passwordConfirmation:='password2');



        //Incorrect minimumValues
        test invalidNewCustomer
             ($mail:='', $phone:='XXXXXXXXX', $company:='UPC', $fax:='XXXXXXXXX',
              $firstName:='Albert', $lastName:='Tort', $street:='Jordi Girona,1',
              $postCode:='08034', $city:='Barcelona', $country:='Espanya', $newsletter:=true,
              $password:='password', $passwordConfirmation:='password');

        test invalidNewCustomer($mail:='', $phone:='XXXXXXXXX', $company:='UPC',
                                $fax:='XXXXXXXXX', $firstName:='Albert', $lastName:='Tort',
                                $street:='Jordi Girona,1', $postCode:='08034',
                                $city:='Barcelona', $country:='Espanya', $newsletter:=true,
                                $password:='pass', $passwordConfirmation:='pass');

        test invalidNewCustomer($mail:='olive@lsi.upc.edu', $phone:='XX', $company:='UPC',
                                $fax:='XXXXXXXXX', $firstName:='Antoni', $lastName:='Olive',
                                $street:='Jordi Girona,1', $postCode:='08034',
                                $city:='Barcelona', $country:='Espanya', $newsletter:=false,
                                $password:='password', $passwordConfirmation:='password');

        test invalidNewCustomer($mail:='atort@lsi.upc.edu', $phone:='XXXXXXXXX',
                                $company:='UPC', $fax:='XXXXXXXXX', $firstName:='Albert',
                                $lastName:='Tort', $street:='', $postCode:='',
                                $city:='Barcelona', $country:='Espanya', $newsletter:=true,
                                $password:='password', $passwordConfirmation:='password');
}
```

```
testprogram EditCustomers{

        textConfigurationValues := new MinimumValues, MaximumValues;
        textConfigurationValues.firstName:=1;
        textConfigurationValues.lastName:=1;
        textConfigurationValues.dateOfBirth:=6;
        textConfigurationValues.eMailAddress:=1;
        textConfigurationValues.streetAddress:=1;
        textConfigurationValues.companyName:=0;
        textConfigurationValues.postCode:=1;
        textConfigurationValues.city:=1;
        textConfigurationValues.state:=1;
        textConfigurationValues.telephoneNumber:=9;
        textConfigurationValues.password:=4;
        textConfigurationValues.addressBookEntries:=2;

        customerDetailsConfiguration :=  new CustomerDetails;
        customerDetailsConfiguration.gender:=false;
        customerDetailsConfiguration.dateOfBirth:=false;
        customerDetailsConfiguration.company:=false;
        customerDetailsConfiguration.state:=false;
        customerDetailsConfiguration.suburb:=false;

        //Customer already created
        e := new EMail(eMail:='john@xxxx.xxx');
        d:= new Date;
        pc:= new PostalCode(postalCode:='XXXXX');
        c := new Country;
        a := new Address(firstName:='John', lastName:='Junior', street:='Major', postCode:=pc,
                         city:='xxxxxxxx', country:=c);

        new NewCustomer(eMailAddress:=e, dateOfBirth:=d, phone:='XXXXXXXXX', fax:='XXXXXXXXX',
                        primary:=a, newsletter:=true, password:='password',
                        passwordConfirmation:='password') occurs;

        john:=Customer.allInstances->any(eMailAddress=e);


        //Password change
        test validPasswordChange{
                new PasswordChange(customer:=john,
                                   oldPassword:='password',
                                   newPassword:='newPassword') occurs;
                assert equals john.password 'newPassword';
        }

        test invalidPasswordChange{
                //The password cannot be changed if the old password is not correct
                new PasswordChange(customer:=john,
                                   oldPassword:='asdfasdf',
                                   newPassword:='newPassword') may not occur;

                //The password cannot be changed if the new password does not satisfies
                //the minimum and maximum configuration values
                new PasswordChange(customer:=john,
                                   oldPassword:='password',
                                   newPassword:='as') may not occur;
        }

        //Edit customer details
        test validCustomerDetailsEditions{
                e2 := new EMail(eMail:='john@yyyyy.yyy');
                d2:= new Date(date:='YY/YY/YYYY');
                new EditCustomerDetails(customer:=john,
                                        newFirstName:='Johnatan', newLastName:='JR.',
                                        newEMailAddress:=e2, newDateOfBirth:=d2,
                                        newPhone:='YYYYYYYYY', newFax:='YYYYYYYYY') occurs;
        }

        test invalidCustomerDetailsEditions{
                e2 := new EMail(eMail:='');
                d2:= new Date(date:='YY/YY');
```

```
                new EditCustomerDetails(customer:=john,
                                        newFirstName:='', newLastName:='',
                                        newEMailAddress:=e2, newDateOfBirth:=d2,
                                        newPhone:='YYYYYY', newFax:='YY') may not occur;
        }

        //Edit customer
        /*Edit customer can only be executed by the store administrator
        (who can edit the customer details including its password and the
        global notifications option*/

        test validCustomerEdition{
                e2 := new EMail(eMail:='john@yyyyy.yyy');
                d2:= new Date(date:='YY/YY/YYYY');
                new EditCustomer(customer:=john,newPassword:='zxcvxcv',
                                 newGlobalNotifications:=false,
                                 newFirstName:='Johnatan', newLastName:='JR.',
                                 newEMailAddress:=e2, newDateOfBirth:=d2,
                                 newPhone:='YYYYYYYYY', newFax:='YYYYYYYYY') occurs;
        }

        test invalidCustomerEdition{
                e2 := new EMail(eMail:='');
                d2:= new Date(date:='YY/YY');
                new EditCustomer(customer:=john,
                                 newPassword:='xy', newGlobalNotifications:=false,
                                 newFirstName:='', newLastName:='', newEMailAddress:=e2,
                                 newDateOfBirth:=d2, newPhone:='YYYYYY', newFax:='YY')
                                 may not occur;
        }
}
```

```
testprogram CustomerAddressesManagement{

        //Customer initialization
        spain:=new Country(name:='Spain', isoCode2:='ES', isoCode3:='ESP');
        catalonia:=new Zone(name:='Catalonia', code:='CAT', country:=spain);
        a:= new Address(country:=spain, zone:=catalonia,
                        state:='Catalonia', street:='Lluis Companys', city:='Sitges');
        c := new Customer(address:=a,primary:=a);

        //Other locations to be used
        germany:=new Country(name:='Germany', isoCode2:='DE', isoCode3:='DEU');
        saxony:=new Zone(name:='Saxony', code:='SAX', country:=germany);
        pc:=new PostalCode(postalCode:='XXXXX');

        //Minimum and maximum values
        textConfigurationValues := new MinimumValues, MaximumValues;
        textConfigurationValues.firstName:=1;
        textConfigurationValues.lastName:=1;
        textConfigurationValues.dateOfBirth:=6;
        textConfigurationValues.eMailAddress:=1;
        textConfigurationValues.streetAddress:=1;
        textConfigurationValues.companyName:=0;
        textConfigurationValues.postCode:=1;
        textConfigurationValues.city:=1;
        textConfigurationValues.state:=1;
        textConfigurationValues.telephoneNumber:=9;
        textConfigurationValues.password:=4;
        textConfigurationValues.addressBookEntries:=2;
        customerDetailsConfiguration :=  new CustomerDetails;
        customerDetailsConfiguration.gender:=false;
        customerDetailsConfiguration.dateOfBirth:=true;
        customerDetailsConfiguration.company:=false;
        customerDetailsConfiguration.state:=false;
        customerDetailsConfiguration.suburb:=false;
```

```
        test validAddressCreations{
                pc:=new PostalCode(postalCode:='XXXXX');
                new NewCustomerAddress(customer:=c, firstName:='XXXX', lastName:='XXXXXX',
                                       street:='XXXXX', postCode:=pc, city:='XXXXX',
                                       country:=spain, zone:=catalonia, state:='Catalonia')
                occurs;
        }

        test invalidAddressCreations{
                //Zone must be coherent with the state if it is assigned
                new NewCustomerAddress(customer:=c, zone:=catalonia, firstName:='XXXX',
                                      lastName:='XXXXXX', street:='XXXXX', postCode:=pc,
                                      city:='XXXXX', country:=spain) may not occur;
                new NewCustomerAddress(customer:=c, zone:=saxony, country:=spain,
                                       firstName:='XXXX', lastName:='XXXXXX', street:='XXXXX',
                                      postCode:=pc, city:='XXXXX') may not occur;

                //Minimum values cannot be violated
                new NewCustomerAddress(customer:=c, zone:=saxony, country:=spain,
                                       firstName:='', lastName:='', street:='XXXXX',
                                       postCode:=pc, city:='') may not occur;
        }

        test AddressEdition{
                //We add to the customer another address
                new NewCustomerAddress(customer:=c, zone:=saxony, country:=germany,
                                       firstName:='XXXXXXXX', lastName:='XXXXXXXX',
                                       street:='XXXXX', postCode:=pc, city:='Dresden',
                                       state:='Saxony') occurs;

                //Now, the customer has addresses in Spain and in Germany
                assert equals c.address.country->asSet() Set{spain,germany};
                assert true c.address->exists(street='Lluis Companys');

                //We try to change the spanish address
                //(we test what if the user lives now in another street)
                //In order to edit an address of a customer we should provide the new address
                na:=new Address(country:=spain, zone:=catalonia, state:='Catalonia',
                                city:='Sitges', street:='Passeig Maritim',
                                postCode:=pc,firstName:='XXXX', lastName:='XXXXXX');

                new EditCustomerAddress(customer:=c, address:=a, newAddress:=na) occurs;
                assert false c.address->exists(street='Lluis Companys');
                assert true c.address->exists(street='Passeig Maritim');

                //We can change the primary address
                //We put the address from Germany as the primary
                new PrimaryCustomerAddressChange(address:=c.address->any(country=germany),
                                                 customer:=c) occurs;

                //We cannot put as primary an address which is not an address of the customer
                a2:= new Address(country:=spain, zone:=catalonia, state:='Catalonia',
                                 street:='Anselm Clavé', city:='Tarragona');
                new PrimaryCustomerAddressChange(address:=a2, customer:=c) may not occur;

                //Minimum values cannot be violated when editing an address
                //We try to edit an address with no city and street information
                na2:=new Address(country:=spain, zone:=catalonia, state:='Catalonia',
                                 city:='', street:='',postCode:=pc,firstName:='XXXX',
                                 lastName:='XXXXXX');
                new EditCustomerAddress(customer:=c, address:=a, newAddress:=na2)
                may not occur;

                //Finally, we delete an address of a customer;
                assert equals c.address->size() 2;
                new DeleteCustomerAddress(address:=c.address->any(country=spain), customer:=c)
                occurs;

                //We cannot delete the primary address
                new DeleteCustomerAddress(address:=c.primary, customer:=c) may not occur;
        }
}
```

```
testcontext ProductSubscriptionsManagement{

        //Customer initialization
        spain:=new Country(name:='Spain', isoCode2:='ES', isoCode3:='ESP');
        catalonia:=new Zone(name:='Catalonia', code:='CAT', country:=spain);
        a:= new Address(country:=spain, zone:=catalonia, state:='Catalonia',
                        street:='Lluis Companys', city:='Sitges');
        c := new Customer(address:=a,primary:=a, globalNotifications:=false);

        //Products initialization
        p1:=new Product;
        p2:=new Product;

        test ProductNotificationSubscriptions{
                assert equals c.notifications()->size() 0;
                new NewProductNotificationSubscription(customer:=c, newSubscribedProduct:=p1)
                occurs;
                assert equals c.notifications() Set{p1};

                //We cannot subscribe an already subscribed product
                new NewProductNotificationSubscription(customer:=c, newSubscribedProduct:=p1)
                may not occur;

                //We can subscribe more than one product
                new NewProductNotificationSubscription(customer:=c, newSubscribedProduct:=p2)
                occurs;
                assert equals c.notifications() Set{p1,p2};

                //We can delete subscriptions
                new DeleteProductNotificationSubscription(customer:=c,
                                                          deletedSubscribedProduct:=p2) occurs;
                assert equals c.notifications() Set{p1};

                //If global notifications is enabled, explicit notification subscriptions
                //are not taken into account and all products are considered to be subscribed
                new EditGlobalNotifications(customer:=c, newGlobalNotifications:=true) occurs;
                assert equals c.notifications() Set{p1,p2};
        }
}
```

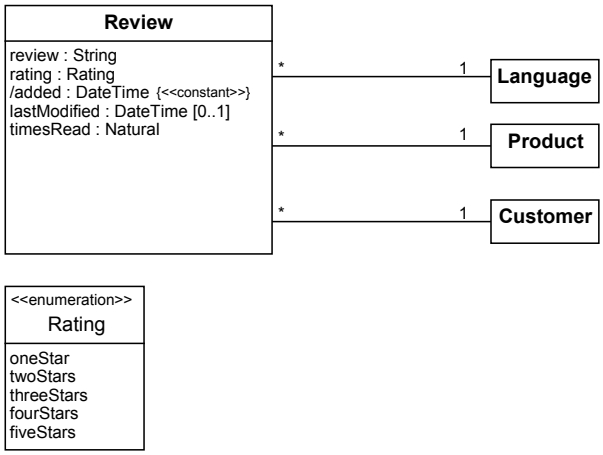
```
testprogram DeleteCustomers{

        //Customer initialization
        co:= new Country;
        a:= new Address(country:=co);
        c:= new Customer(address:=a, primary:=a);
        cu:=new Currency(status:=#enabled);

        //Language initialization
        l:= new Language;

        //Products initialization
        p1:=new Product;
        p2:=new Product;

        //MinimumValues
        mv:=new MinimumValues;
        mv.reviewText:=0;

        //The customer wrote reviews
        new NewReview(customer:=c, product:=p1, language:=l, rating:=#fourStars,
                      review:='reviewText') occurs;
        new NewReview(customer:=c, product:=p2, language:=l, rating:=#twoStars,
                      review:='reviewText2') occurs;

        //The customer has an active shopping cart
        sc := new CustomerShoppingCart(customer:=c);
```

```
        item1 := new ShoppingCartItem(product:=p1, quantity:=3, shoppingCart:=sc);
        test deleteCustomerWithNoOrders{
                //The customer is deleted and also its active shopping carts and reviews
                new DeleteCustomer(customer:=c) occurs;

                //Reviews of customer are also deleted
                assert equals p1.review->size() 0;
                assert equals p2.review->size() 0;

                //The active shopping cart of the customer is also deleted
                assert true c.customerShoppingCart->isEmpty();
        }

        test deleteCustomerWithOrders{
                //Store initialization
                s:=new Store;
                s.defaultLanguage:=l;
                s.defaultCurrency:=cu;
                s.country:=co;
                cos:=new OrderStatus;
                cosl:=new OrderStatusInLanguage(language:=l,orderStatus:=cos);
                cosl.name:='cancelled';
                s.cancelledStatus:=cos;
                dos:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=l);
                dosl.name:='pending';
                s.defaultStatus:=dos;

                //We create an order of the customer
                stock := new Stock;
                stock.checkStockLevel:=false;
                stock.allowCheckout:=true;
                stock.substractStock:=false;

                pm:=new CashOnDelivery(status:=#enabled);
                sm:=new PerItem(status:=#enabled, handlingFee:=5, cost:=10);

                new OrderConfirmation(shoppingCart:=sc, currency:=cu ,
                                      shippingMethod:=sm, paymentMethod:=pm) occurs;


                new DeleteCustomer(customer:=c) occurs;

                //The customer becomes disabled and also its active shopping carts and reviews
                assert equals c.status #disabled;

                //Reviews of customer are also deleted
                assert equals p1.review->size() 0;
                assert equals p2.review->size() 0;

                //The active shopping cart of the customer is also deleted
                assert true c.customerShoppingCart->isEmpty();
        }
}
```

9.17. Reviews

Structural schema

In order to allow users reading evaluations of a product, customers can write reviews.



[1] *Review::**added*** is the *DateTime* of the review creation.

```
context Review::added():DateTime
  body : Now()
```

Use cases

Add a review

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to write a review of a product.

Main Success Scenario:

1. The customer selects a product.
2. The customer provides the content and the rate of the review:

   [→*NewReview*]
3. The system validates that the data is correct.
4. The system saves the review.

Extensions:

2a. The customer is not logged in:

2a1. The customer logs in:

[→*LogIn*]

2a2. The use case continues at step 2.

Edit a review

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a review.

Main Success Scenario:

1. The store administrator selects the review to be edited.
2. The store administrator provides the modified text and the new rating of the selected review.

   [→*EditReview*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete a review

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a review.

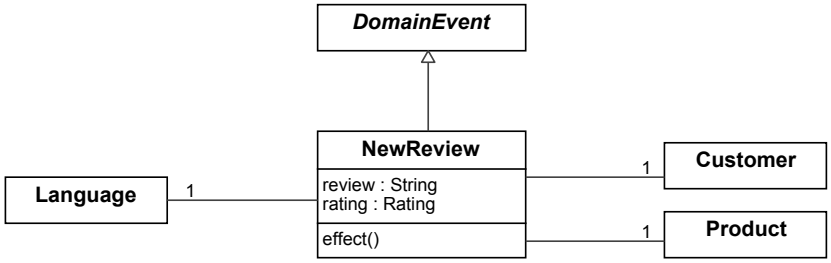
Main Success Scenario:

1. The store administrator selects the review to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the review:

   [→*DeleteReview*]
4. The system deletes the review.

Events

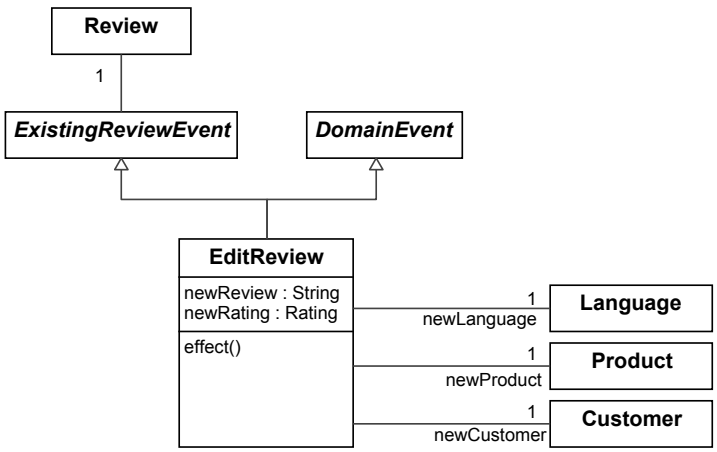
NewReview



«IniIC»
context NewReview::reviewRight(): Boolean
body : self.review.size() >= MinimumValues.reviewText

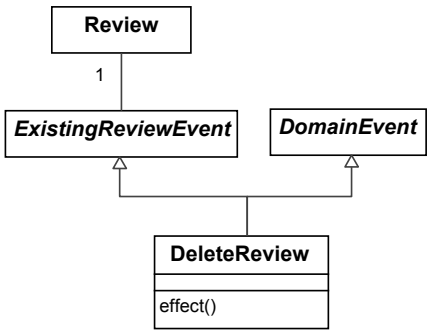
context NewReview::effect()
post :
r.oclIsNew() **and**
r.oclIsTypeOf(Review) **and**
r.review = self.review **and**
r.rating = self.rating **and**
r.customer = self.customer **and**
r.product = self.product **and**
r.language = self.language

EditReview



```
context EditReview::effect()
  post :
     self.review.review = self.newReview and
     self.review.rating = self.newRating and
     self.review.language = self.newLanguage and
     self.review.product = self.newProduct and
     self.review.customer = self.newCustomer
  post :
     self.review.lastModified = Now()
```

DeleteReview




```
context DeleteReview::effect()
  post : not self.review@pre.oclIsKindOf(OclAny)
```

Example test programs

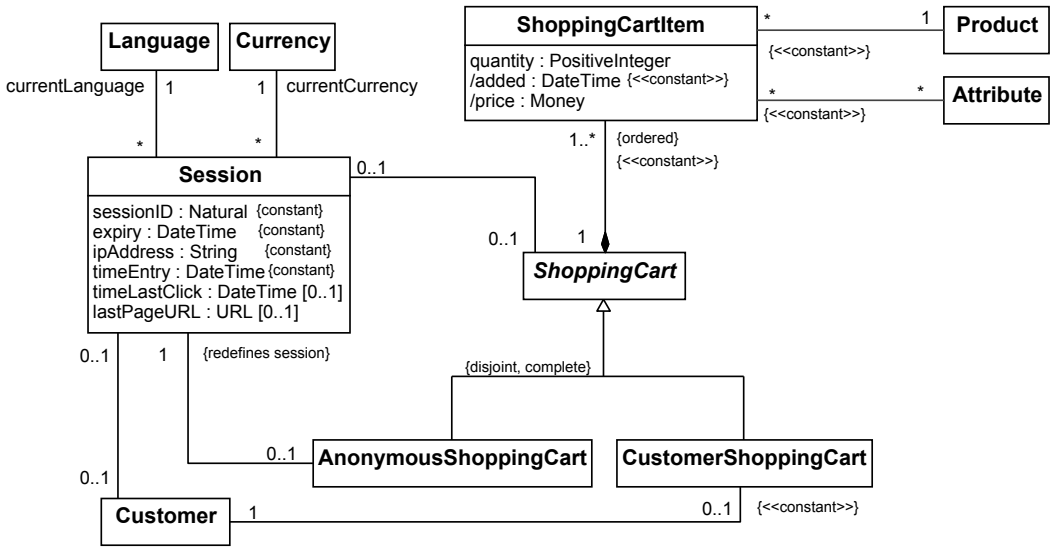
```
testprogram ReviewsManagement{

                english:=new Language(name:='English', code:='EN');
                spanish:=new Language(name:='Spanish', code:='ES');
                usa:=new Country;
                a1:= new Address(country:=usa);
                e1:= new EMail(eMail:='xxxx1@x.com');
                c1:=new Customer(eMailAddress:=e1,address:=a1,primary:=a1);
                a2:= new Address(country:=usa);
                e2:= new EMail(eMail:='xxxx2@x.com');
                c2:=new Customer(eMailAddress:=e2,address:=a2,primary:=a2);
                hotelcomfort:=new Product;

                new MinimumValues(reviewText:=1);

                test newReview{
                        new NewReview(customer:=c1, product:=hotelcomfort,
                                      language:=english, rating:=#fourStars,
                                       review:='Very easy to find the hotel near Notting Hill
                                                gate. Generally very polite and helpful people
                                                in the area') occurs;
                }

                test ThreeReviewsOfProduct{
                        new NewReview(customer:=c1, product:=hotelcomfort,
                                      language:=english, rating:=#fourStars,
                                       review:='Very easy to find the hotel near Notting Hill
                                                gate. Generally very polite and helpful people
                                                in the area') occurs;
```

```
                        new NewReview(customer:=c2, product:=hotelcomfort,
                                    language:=spanish, rating:=#twoStars,
                                     review:='Muy bien localizado, al lado del mercado de
                                             Porto Bello. És un hotel con una distribución
                                             estraña al ocupar varios edificios lo que hace
                                             que el laberinto de pasillos sea de lo más
                                             divertido. El personal es distante.') occurs;

                        //A customer can review a product more than once
                        new NewReview(customer:=c1, product:=hotelcomfort,
                                    language:=english, rating:=#fourStars,
                                     review:='Easy accessible by public transport') occurs;

                        assert equals hotelcomfort.review->size() 3;
                }

                test InvalidReviewCreation{
                        //Minimum values configuration must be taken into account
                        new NewReview(customer:=c1, product:=hotelcomfort,
                                    language:=english, rating:=#fourStars,
                                     review:='') may not occur;
                }

                test ReviewEdition{
                        //A customer can publish a review
                        new NewReview(customer:=c1, product:=hotelcomfort,
                                    language:=english, rating:=#fiveStars,
                                     review:='I hate this hotel. Call me for more
                                    details 12345') occurs;

                        //And the store administrator can edit it
                        new EditReview(review:=nr.createdReview, newLanguage:=english,
                                     newCustomer:=c1, newRating:=#oneStar,
                                     newProduct:=hotelcomfort,
                                     newReview:='I do not like this hotel') occurs;
                }

                test DeleteReview{
                        //A customer can publish a review
                        nr:=new NewReview(customer:=c1, product:=hotelcomfort,
                                        language:=english, rating:=#fiveStars,
                                         review:='asdfasdfñjñasdf');

                        assert equals hotelcomfort.review->size() 1;

                        //And the store administrator can delete it
                        r:=nr.createdReview;
                        new DeleteReview(review:=r) occurs;

                        assert equals hotelcomfort.review->size() 0;
                }


}
```

9.18. Shopping carts & Orders

Structural schema

Customers can add or remove products from their shopping carts while they are surfing the *online* store.



[DR1] *ShoppingCartItem::**price*** is the net price for an item taking into account the selected product attributes.

```
context ShoppingCartItem::price():Money
body :
   let netPriceWithSpecial:Money =
            if self.product.specialNetPrice ->notEmpty() then self.product.specialNetPrice
            else self.product.netPrice
            endif
   in
   if  self.attribute -> isEmpty() then netPriceWithSpecial
   else
      self.attribute.productAttribute -> select (pa | pa.product = self.product) -> collect
      (if sign = Sign::plus
      then increment
       else –increment
      endif) -> sum() + netPriceWithSpecial
   endif
```

[DR2] *ShoppingCartItem::**added*** is the *DateTime* when the item was created.

```
context ShoppingCartItem::added():DateTime
  body : Now()
```

[IC1] 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

[IC2] The shopping cart item specifies the selected product attributes, which must be a subset of all the product attributes.

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

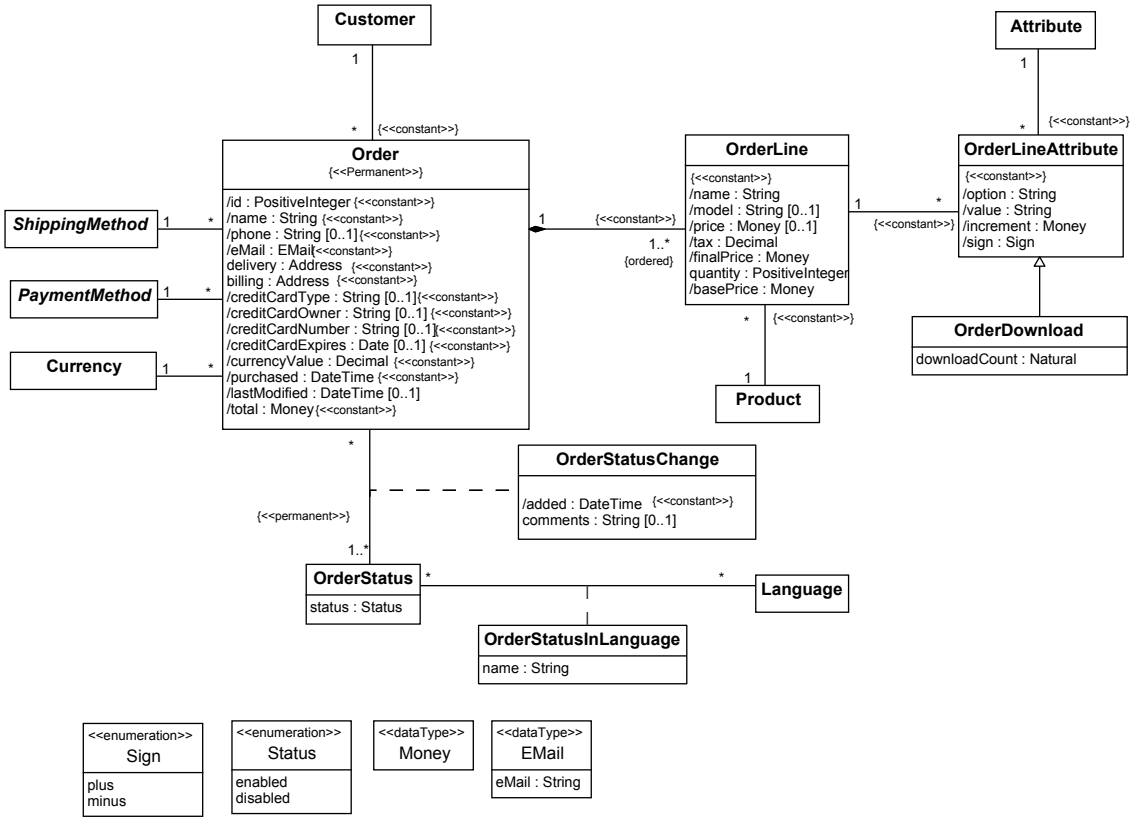
[IC3] The shopping cart item specifies only one attribute per option.

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

[IC4] Sessions are identified by its sessionID.

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

Orders are the confirmation that a customer wants to buy the contents of his shopping cart.



```
context ShippingMethod def:
  addTaxes(z:Zone, basePrice:Money) : Money =
    let appliedTaxRates:Set(TaxRate)=
      z.taxZone.taxRate -> select (tr | tr.taxClass = self.taxClass) -> asSet()
    in
      let priorities:set(Natural) =
        if appliedTaxRates -> isEmpty() then set{}
        else appliedTaxRates -> sortedBy(priority).priority -> asSet()
        endif
      in
        if priorities -> isEmpty() then basePrice
        else priorities -> iterate (p:Natural; res:Money = 0 |
              res +
              (((appliedTaxRates -> select (tr | tr.priority = p).rate
              -> sum()) / 100)+1)*basePrice)
        endif

context ShippingMethod def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money = 0

context FlatRate def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money = self.cost

context PerItem def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money =
    self.cost*quantity

context TableRate def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money =
    if self.method = ShippingTableMethod::weight
    then
      self.items -> select (i | i.number <= (totalWeight*quantity)) -> sortedBy(number) ->last().cost
    else
      self.items -> select (i | i.number <= (totalPrice*quantity)) -> sortedBy(number) ->last().cost
    endif

context USPostalService def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money =
    calculateFromUSPS (self.userID, self.password, self.server, totalWeight, totalPrice, quantity)
```

[DR1] *Order::**id*** identifies the order and it is assigned automatically.

```
context Order::id():PositiveInteger
body :
  if Order.allInstances() -> size() = 0 then 0
  else Order.allInstances() -> sortedBy(id) -> last().id + 1
  endif
```

[DR2] *Order::**primary*** address of an order is that of its customer.

```
context Order::primary():Address
 body : self.customer.primary
```

[DR3] *Order::**eMailAddress*** of an order is that of its customer.

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

[DR4] *Order::**phone*** of an order is that of its customer.

```
context Order::phone():String
  body : self.customer.phone
```

[DR5] *Order::**purchased*** is the *DateTime* when the order was created

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

[DR6] *Order::**lastModified*** is the last *DateTime* when the status order was modified

```
context Order::lastModified():DateTime
  body : self.orderStatusChange -> sortedBy(added)  -> last().added
```

[DR7] *Order::**statuts*** is the current status of the order

```
context Order::status():OrderStatus
  body : self.orderStatusChange -> sortedBy(added) -> last().orderStatus
```

[DR8] *Order::**total*** gives the total amount of an order

```
context Order::total():Money
body :
   let totalWithoutShippingCosts:Money =
      self.orderLine -> collect(finalPrice*quantity) -> sum()
   let totalWeight:Decimal =
      self.orderLine -> collect(product.weight*quantity) -> sum()
   let quantity:PositiveInteger =
      self.orderLine.quantity -> sum()
   let handlingFee:Money =
      if self.shippingMethod.oclIsKindOf(HandlingFeeMethod)
      then
         self.shippingMethod.oclAsType(HandlingFeeMethod).handlingFee
      else 0
      endif
   in
      let totalWeightIncreased:Decimal =
            if totalWeight* (ShippingAndPackaging.percentageIncreaseForLargerPackages/100) >
               ShippingAndPackaging.typicalPackageTareWeight
            then
             totalWeight * (1 +totalWeight*
             ShippingAndPackaging.percentageIncreaseForLargerPackages/100)
         else totalWeight + ShippingAndPackaging.typicalPackageTareWeight
         endif
       in
         totalWithoutShippingCosts +
               self.shippingMethod.shippingCosts
               (totalWeightIncreased, totalWithoutShippingCosts, quantity) + handlingFee
```

[DR9] *OrderStatusChange::**added*** is the *DateTime* when the change is done.

```
context OrderStatusChange::added():DateTime
body :  Now()
```

[10] *OrderLine::**name*** is that of its product in the default language

```
context OrderLine::name():String
body :
   self.product.productInLanguage
   ->select(pil | pil.language = Store.allInstances() -> any(true).defaultLanguage).name
```

[DR11] *OrderLine::**model*** is that of its product

```
context OrderLine::model():String
body : self.product.model
```

[DR12] *OrderLine::**basePrice*** is the net price of the product without taking into account the selected attributes.

```
context OrderLine::basePrice():Money
body :
   if self.product.specialNetPrice ->notEmpty()
   then self.product.specialNetPrice
   else self.product.netPrice
   endif
```

[DR13] *OrderLine::**price*** is the net price of the product with the selected attributes

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

[DR14] *OrderLine::**finalPrice*** is the price of the product with the selected attributes and taking into account the taxes

```
context OrderLine::finalPrice():Money
body :
   if self.billing.zone -> notEmpty() then
      self.product.addTaxes(self.billing.zone, self.price)
   else self.price
   endif
```

[DR15] *OrderLineAttribute::**option*** is the option name in the default language

```
context OrderLineAttribute::option():String
body :
   self.attribute.option.hasOptionName
      -> select (hon | hon.optionLanguage = Store.allInstances()  -> any(true).defaultLanguage).optionName
```

[DR16] *OrderLineAttribute::**value*** is the option value in the default language

```
context OrderLineAttribute::value():String
body :
   self.attribute.value.hasValueName
      -> select (hvn | hon.valueLanguage = Store.allInstances() -> any(true).defaultLanguage).valueName
```

[DR17] *OrderLineAttribute::**increment*** is the increment applied in the product price by the attribute

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

[DR18] *OrderLineAttribute::**sign*** is the sign of the increment applied in the product price by the attribute

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

[IC1] A specific zone shipping method with a specific tax zone can only be applied if the delivery address zone is included in the tax zone.

```
context Order::ApplicableZoneShippingMethod: Boolean
body :
   self.shippingMethod.oclIsTypeOf(SpecificZoneMethod) and
   self.shippingMethod.oclAsType(SpecificZoneMethod).taxZone -> notEmpty implies
   self.shippingMethod.oclAsType(SpecificZoneMethod).taxZone.zone
   -> includes(self.delivery.zone)
```

[IC2] The *Zone Rates* shipping method can only be applied in the specified countries.

```
context Order::ApplicableZoneRatesShippingMethod: Boolean
body :
   self.shippingMethod.oclIsTypeOf(ZoneRates)  implies
   self.shippingMethod.oclAsType(ZoneRates).country -> includes(self.delivery.country)
```

[IC3] Payment methods with a specified tax zone can only be applied in orders with a billing address located in a zone included in the tax zone.

```
context Order::ApplicableZonesPaymentMethod: Boolean
body :
   self.paymentMethod.taxZone -> notEmpty()  implies
   self.paymentMethod.taxZone.zone -> includes(self.billing.zone)
```

[IC4] Payment methods with a specified set of applicable currencies can only be applied if the current currency is included in that set.

```
context Order::ApplicableCurrenciesPaymentMethod: Boolean
body :
   self.shippingMethod.oclIsTypeOf(SpecificCurrenciesMethod)  implies
   self.shippingMethod.oclAsType(SpecificCurrenciesMethod).currency -> includes(self.currency)
```

[IC5] Orders are identified by its id

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

[IC6] Order status are identified by its name

```
context OrderStatus::NameIsUnique: Boolean
body :  OrderStatus.allInstances() -> isUnique(name)
```

Use Cases

Open session

Primary Actor: Customer

Precondition: None.

Trigger: A customer starts using the system.

Main Success Scenario:

1. The system creates an anonymous session :

   [→*NewSession*]

Finish session

Primary Actor: Customer

Precondition: None.

Trigger: A customer finishes using the system.

Main Success Scenario:

1. The system deletes the current session.

   [→*DeleteSession*]

Extensions:

1a. The customer is logged in and the session has a non empty shopping cart.

   1a1. The shopping cart is saved.

Log in

Primary Actor: Customer

Precondition: The customer is not logged in yet.

Trigger: A customer logs in the system.

Main Success Scenario:

1. The customer introduces their identification data.
2. The system validates the identification data.

3. The customer becomes the owner of the current session.

[→*LogIn*]

Extensions:

3a. The customer has a shopping cart from a previous session.

3a1. The previous shopping cart is restored.

[→*RestorePreviousShoppingCart*]

3b. The current session has a non-empty and anonymous shopping cart

3b1. The anonymous shopping cart becomes the current shopping cart of the customer.

LogOut

Primary Actor: Customer

Precondition: The customer is logged in.

Trigger: A customer logs out from the system.

Main Success Scenario:

1. The current session becomes anonymous.

[→*LogOut*]

Extensions:

1a. The customer has a non empty shopping cart.

1a1. The shopping cart is saved.

Change the current language

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to change the current language of the session.

Main Success Scenario:

1. The store administrator selects the language which will become the current language.
2. The system updates the current language.

[→*SetCurrentLanguage*]

Change the current currency

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to change the current currency of the session.

Main Success Scenario:

1. The store administrator selects the currency which will become the current currency.
2. The system updates the current currency.

   [→*SetCurrentCurrency*]

Place and order

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to place and order.

Main Success Scenario:

1. At any time before step 10 the customer logs in:

   [→*LogIn*]

The system adds the contents of the anonymous shopping cart to the customer shopping cart.

2. The system displays the contents of the shopping cart.
3. The customer browses the product catalog.

   [→*ReadProductInfo*]

4. The customer selects a product to buy:

   [→*AddProductToShoppingCart*]

5. The system adds the product to the shopping cart.
6. The system displays the contents of the shopping cart.
7. The customer changes the contents of the shopping cart:

   [→*UpdateShoppingCart*]

8. The system updates the shopping cart.
9. The system displays the contents of the updated shopping cart.

   The customer repeats steps 3,4 and 7 as necessary to build his order.

10. The customer checks out the order.
11. The system shows the shipping address and the available shipping methods.
12. The customer selects the preferred shipping method.

13. The system shows the billing address and the available payment methods.
14. The customer selects the preferred payment method.
15. The system displays a summary of the order.
16. The customer confirms the order:

    [→*OrderConfirmation*]

17. The system saves the order.
18. The system sends an email to the customer and to the store extra order emails with the information about the order.

Extensions:

1a. The customer is new:

1a1. Create customer.

5a. The configurable option *Display cart after adding a product* is disabled

The customer repeats steps 3 and 4 as necessary.

5a1. The customer continues with the checkout procedure at step 9.

16a. The customer wants to change the contents of the shopping cart:

16a1. The customer changes the contents of the shopping cart:

[→*UpdateShoppingCart*]

16a2. The customer continues with the checkout procedure at step 11.

11a, 16a. The customer wants to change the shipping address:

11a1. The system shows the know addresses of the customer.

11a2. The customer selects a different shipping address.

11a3. The customer continues with the checkout procedure at step 11.

13a, 16b. The customer wants to change the billing address:

13a1. The system shows the know addresses of the customer.

13a2. The customer selects a different billing address.

13a3. The customer continues with the checkout procedure at step 13.

16c. The customer wants to change the shipping method:

16c1. The customer selects the new shipping method.

16c2. The customer continues with the checkout procedure at step 13.

16d. The customer wants to change the payment method:

16d1. The customer selects the new payment method.

16d2. The customer continues with the checkout procedure at step 15.

11a2a,16a2a. The customer wants to define a new shipping address:

11a2a1. The customer gives the new address:

[→*NewCustomerAddress*]

11a2a2. The system saves the address.

11a2a3. The customer continues with the checkout procedure at step 11.

13a2a,16b2a. The customer wants to define a new billing address:

13a2a1. The customer gives the new address:

[→*NewCustomerAddress*]

13a2a2. The system saves the address.

13a2a3. The customer continues with the checkout procedure at step 13.

Cancel an order

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to cancel an order.

Main Success Scenario:

1. The store administrator selects the order to be cancelled.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to cancel the order:

   [→*CancelOrder*]
4. The system sets the order status to cancelled.

Add an order status

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new order status.

Main Success Scenario:

1. The store administrator provides the details of the new order status:

   [→*NewOrderStatus*]
2. The system validates that the data is correct.
3. The system saves the new order status.

Edit an order status

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit an order status.

Main Success Scenario:

1. The store administrator selects the order status to be edited.
2. The store administrator provides the new details of the selected order status:

   [→*EditOrderStatus*]
3. The system validates that the data is correct.
4. The system saves the changes.

Delete an order status

Primary Actor: Store administrator

Precondition: The deleted order status is not the current status of any order.

Trigger: The store administrator wants to delete an order status.

Main Success Scenario:

1. The store administrator selects the order status to be deleted.
2. The store administrator confirms that he wants to delete the order status:

   [→*DeleteOrderStatus*]
3. The system deletes the order status.

Extensions:

2a. The order status has been an status of an order:

   2a1. The system changes the status of the order status to disabled.

   2a2. The use case ends.

Change the status of an order

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the status of an order.

Main Success Scenario:

1. The system shows the orders and their status.
2. The store administrator selects the order which will be edited.
3. The system shows the applicable order status.
4. The store administrator selects the new status.

[→*UpdateOrderStatus*]

5. The system validates that the data is correct.
6. The system saves the changes.

Set cancelled order status

Primary Actor: Store administrator

Precondition: The order status is not yet the cancelled status.

Trigger: The store administrator wants to indicate to the system which order status is used to indicate that an order is cancelled.

Main Success Scenario:

1. The store administrator selects an order status.
2. The system register that the selected order status represents cancelled orders.

   [→*SetCancelledOrderStatus*]

Set default order status

Primary Actor: Store administrator

Precondition: The order status is not yet the default status.

Trigger: The store administrator wants to indicate to the system which order status is assign when an order is created.

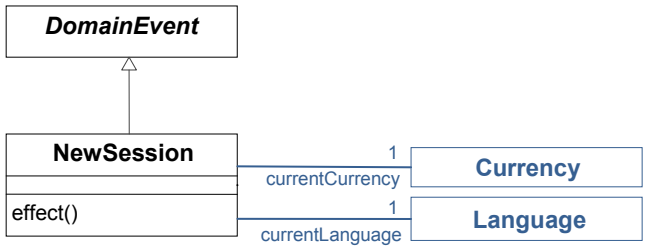
Main Success Scenario:

1. The store administrator selects an order status.
2. The system register that the selected order status is the default order status.

   [→*SetDefaultOrderStatus*]

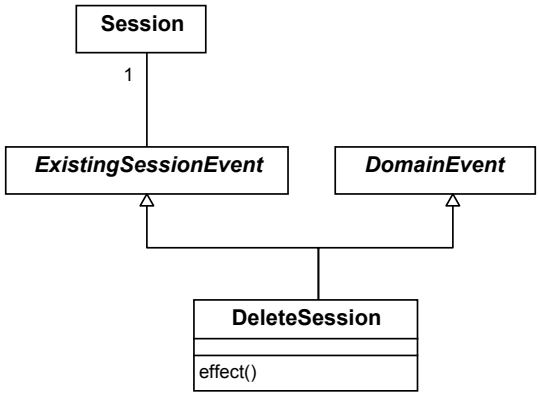
Events

NewSession



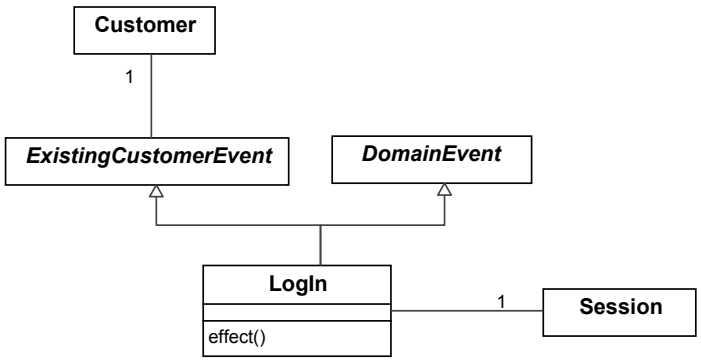

```
context  NewSession::effect()
  post :
     s.oclIsNew() and
     s.oclIsTypeOf(Session) and
     s.currentCurrency=self.currentCurrency and
     s.currentLanguage=self.currentLanguage and
     s.sessionID=Session.allInstances->size()
```

DeleteSession




```
context DeleteSession::effect()
  post : not self.session@pre.oclIsKindOf(OclAny)
```

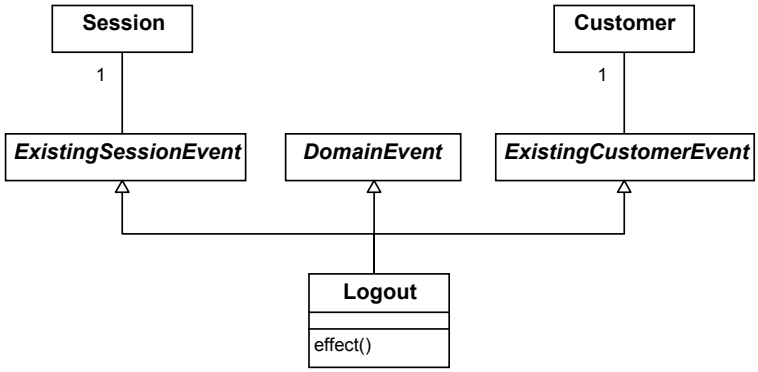
LogIn



«IniIC»
context LogIn::customerIsNotLoggedIn (): Boolean
body : self.customer.session -> isEmpty()

context LogIn::effect()
post :
self.session.customer = self.customer
post :
self.customer.numberOfLogons = self.customer.numberOfLogons@pre + 1
post:
if self.customer.customerShoppingCart->size()>0 **then**
rpsc.oclIsNew() **and**
rpsc.oclIsTypeOf(RestorePreviousShoppingCart) **and**
rpsc.customer=self.customer **and**
rpsc.session=self.session
else
if self.session.shoppingCart->notEmpty() **then**
csc.oclIsNew() and
csc.oclIsTypeOf(CustomerShoppingCart) and
csc.shoppingCartItem = self.session.shoppingCart.shoppingCartItem and
csc.customer=self.customer and
self.session.shoppingCart=csc
else true
endif
endif

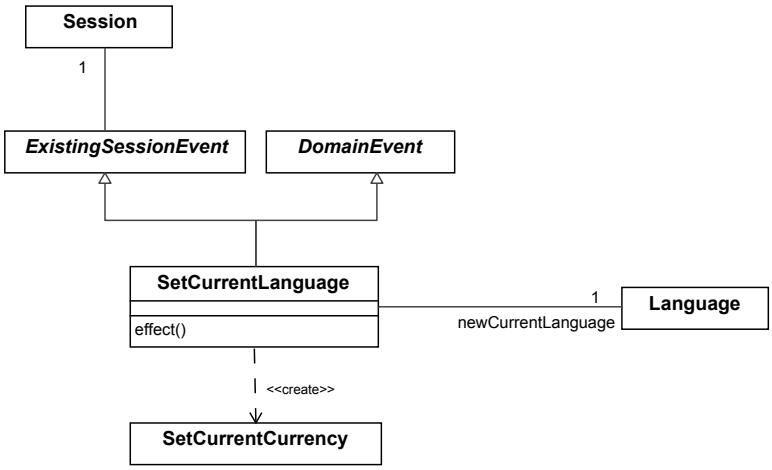
LogOut



«IniIC»
context LogOut::customerIsLoggedIn (): Boolean
body : self.session.customer = self.customer

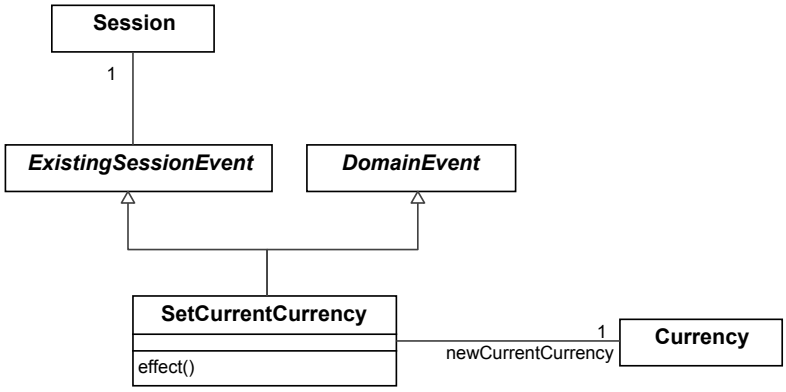
context LogOut::effect()
post : self.session.customer -> isEmpty()

SetCurrentLanguage



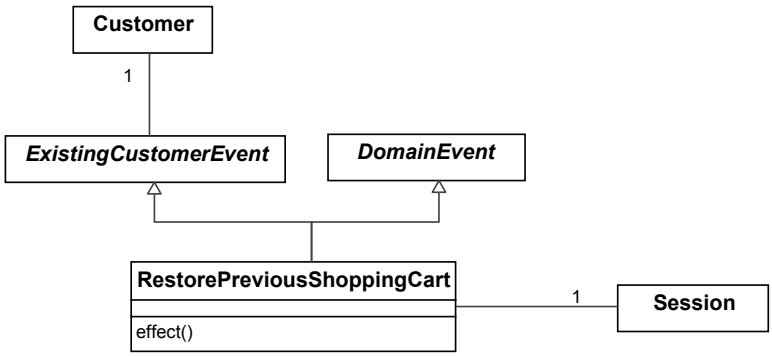
context ChangeCurrentLanguage::effect()
post :
session.currentLanguage = self.newCurrentLanguage
post :
Store.allInstances() -> any(true).switchToDefaultLanguageCurrency **and**
self.newCurrentLanguage.defaultCurrency -> notEmpty()
implies
ccc.oclIsNew() **and**
ccc.oclIsTypeOf(ChangeCurrentCurrency) **and**
ccc.session = self.session **and**
ccc.newCurrentCurrency = self.language.defaultCurrency

SetCurrentCurrency



context SetCurrentCurrency::effect()
post : self.session.currentCurrency = self.newCurrentCurrency

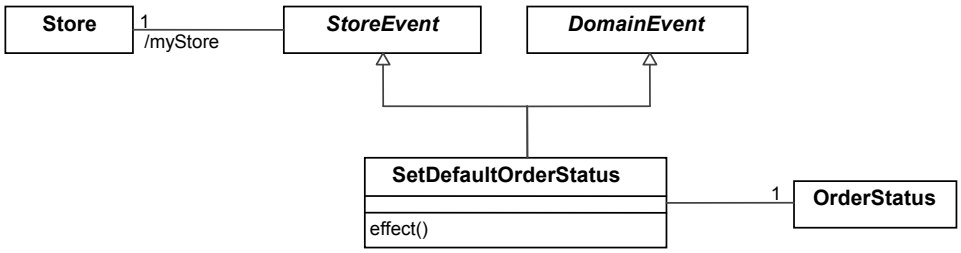
RestorePreviousShoppingCart



«IniIC»
context RestorePreviousShoppingCart::CustomerHasAPreviousShoppingCart(): Boolean
body : self.customer.customerShoppingCart->notEmpty()

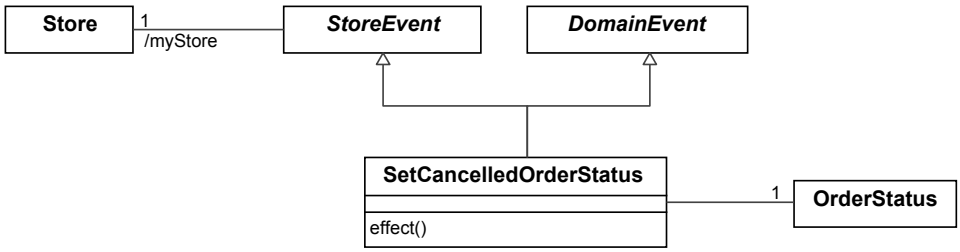
context RestorePreviousShoppingCart::effect()
post : self.session.shoppingCart = self.customer.customerShoppingCart

SetDefaultOrderStatus



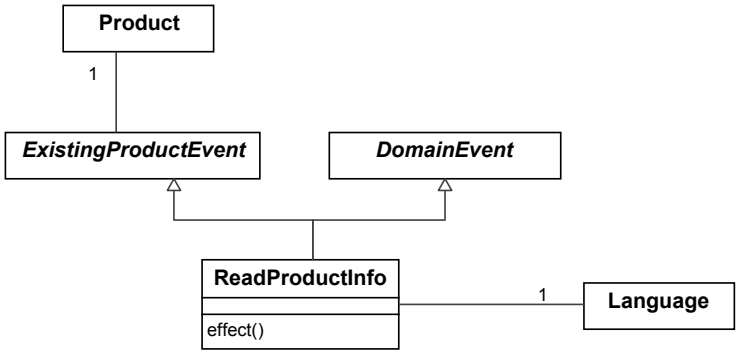
context SetPendingOrderStatus::effect()
post : self.myStore.defaultStatus = self.orderStatus

SetCancelledOrderStatus



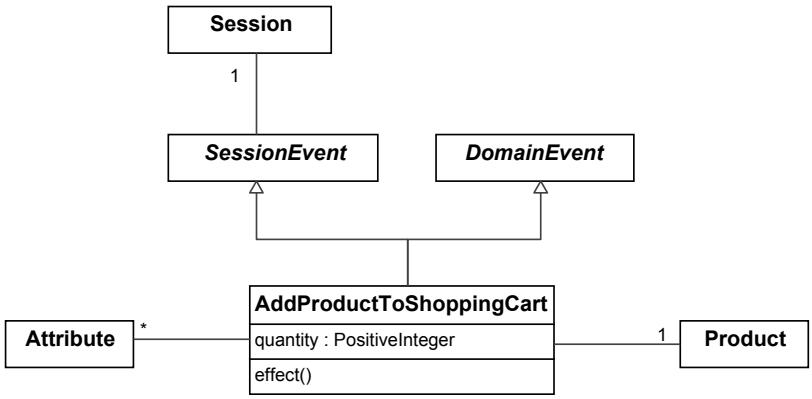
context SetCancelledOrderStatus::effect()
post : self.myStore.cancelledStatus = self.orderStatus

ReadProductInfo



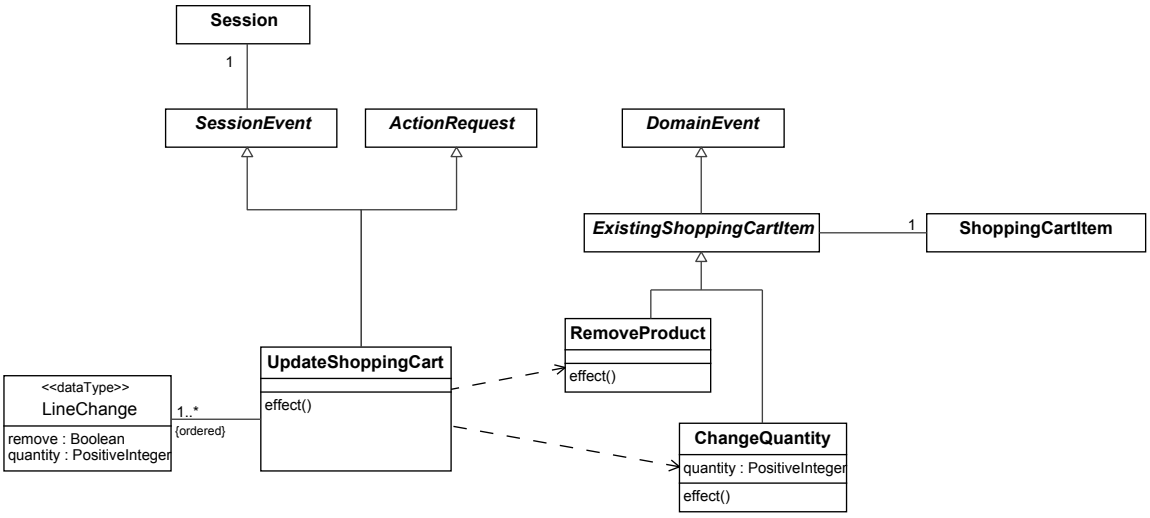
context ReadProductInfo::effect()
post : self.product.productInLanguage->select(pil | pil.language=self.language).viewed =
self.product@pre.productInLanguage@pre->select(pil | pil.language=self.language).viewed + 1

AddProductToShoppingCart




```
«IniIC»
context AddProductToShoppingCart::AttributesAreFromProduct(): Boolean
  body : self.product.attribute -> includesAll(self.attribute)

«IniIC»
context AddProductToShoppingCart::AttributesAreOfDifferentOptions(): Boolean
  body : self.attribute -> isUnique(option)

context  AddProductToShoppingCart::effect()
   post ShoppingCartItemIsCreated :
      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
            sc.oclIsNew()  and
            sc.oclIsTypeOf(AnonymousShoppingCart)  and
            self.session.shoppingCart = sc and
            sc.shoppingCartItem -> includes(sci)
        else
         --The customer has logged in
            if self.session.customer.customerShoppingCart -> notEmpty()  then
               --The customer has a previous shopping cart
              self.session.shoppingCart.shoppingCartItem -> includes(sci)
            else
               --The customer does not have a previous shopping cart
               csc.oclIsNew()  and
               csc.oclIsTypeOf(CustomerShoppingCart)  and
               self.session.shoppingCart = csc and
               csc.shoppingCartItem -> includes(sci)
           endif
        endif
      endif
```

UpdateShoppingCart



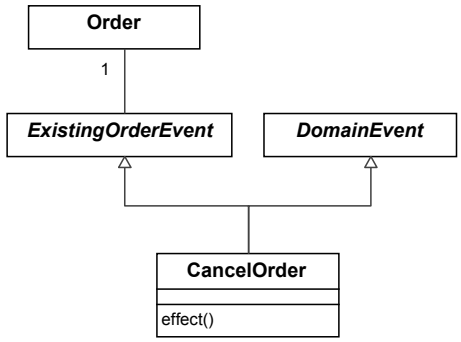
«IniIC»
context UpdateShoppingCart::complete(): Boolean
body : self.lineChange->size() = self.session.shoppingCart.shoppingCartItem->size()

context RemoveProduct::effect()
post : **not** self.shoppingCartItem@pre.oclIsKindOf(OclAny)

context ChangeQuantity::effect()
post : self.shoppingCartItem.quantity = self.quantity

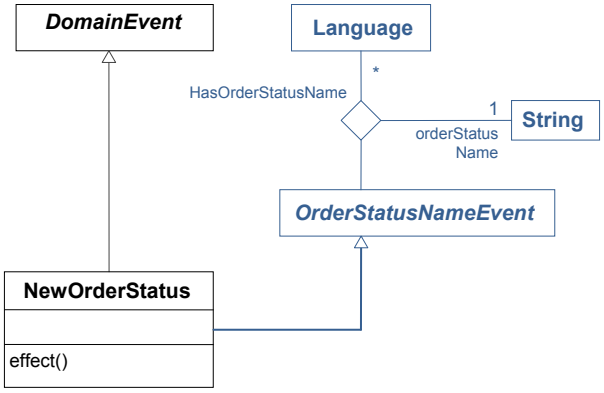
```
context UpdateShoppingCart::effect()
post :
   self.lineChange ->forAll
     (lc | let cartItem:ShoppingCartItem =
                self.shoppingCart.shoppingCartItem->
                at(lineChange->indexOf(lc))
          in
              (lc.remove or lc.quantity <> cartItem.quantity)
                 implies
                    if lc.remove then
                      rp.oclIsNew and
                      rp.oclIsTypeOf(RemoveProduct) and
                      rp.shoppingCartItem = cartItem
                    else
                      cq.oclIsNew() and
                      cq.oclIsTypeOf(ChangeQuantity) and
                      cq.shoppingCartItem = cartItem and
                      cq.quantity = quantity
                    endif )
```

CancelOrder



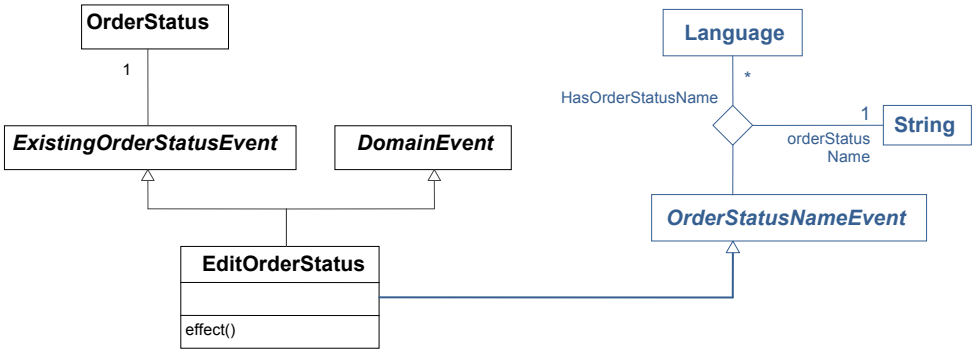

```
context CancelOrder::effect()
  post:
      self.order.orderStatusChange -> sortedBy(added) -> last().orderStatus =
      Store.allInstances() ->any(true).cancelledStatus
```

NewOrderStatus




```
«IniIC»
context NewOrderStatus::orderStatusDoesNotExist(): Boolean
  body :
      not OrderStatus.allInstances -> exists (os |
         Language.allInstances->
          exists(l|
               self.hasOrderStatusName->select(languageOfOrderStatus=l).orderStatusName =
               os.orderStatusInLanguage-> select(language=l).name))

context  NewOrderStatus::effect()
  post :
     os.oclIsNew()  and
     os.oclIsTypeOf(OrderStatus) and
     Language.allInstances->
      forAll(l|
           self.hasOrderStatusName->select(languageOfOrderStatus=l).orderStatusName.string=
           os.orderStatusInLanguage->select(language=l).name)
```

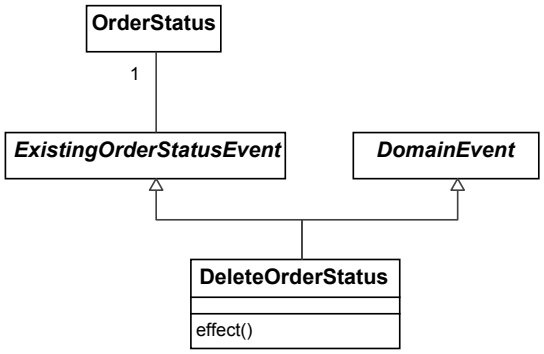
EditOrderStatus



«IniIC»

```
context EditOrderStatus::orderStatusDoesNotExist():Boolean
  body:
    Language.allInstances -> forAll ( l |
         l.orderStatusInLanguage.name
            ->excludes(self.hasOrderStatusName -> any(languageOfOrderStatus=l).orderStatusName)
        or
        l.orderStatusInLanguage->any(orderStatus=self.orderStatus).name =
        self.hasOrderStatusName->any(languageOfOrderStatus=l).orderStatusName)

context  EditOrderStatus::effect()
  post :
    Language.allInstances ->  forAll(l|
          self.hasOrderStatusName->select(languageOfOrderStatus=l).orderStatusName =
          self.orderStatus.orderStatusInLanguage->
              select(language=l).name)
```

DeleteOrderStatus



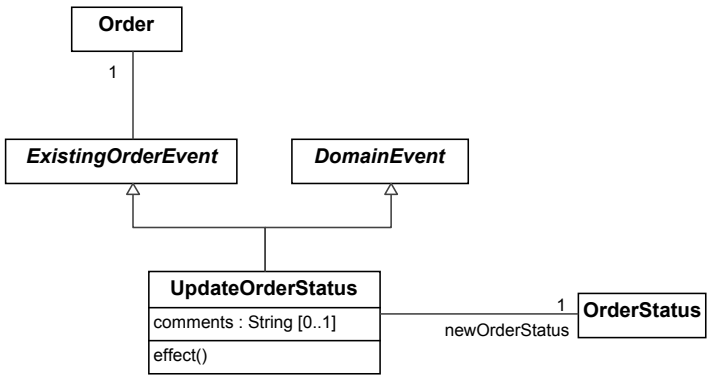
«IniIC»

```
context DeleteOrderStatus:: IsNotTheCurrentStatusOfAnyOrder(): Boolean
  body :
    Order.allInstances() -> forAll (o | o.orderStatusChange -> sortedBy(added)
                                        -> last().orderStatus <> self.orderStatus)
```

«IniIC»
context DeleteOrderStatus::IsNotADefaultStatus():Boolean
body:
Store.allInstances->forAll(s| s.defaultStatus <> self.orderStatus **and** s.cancelledStatus <> self.orderStatus)

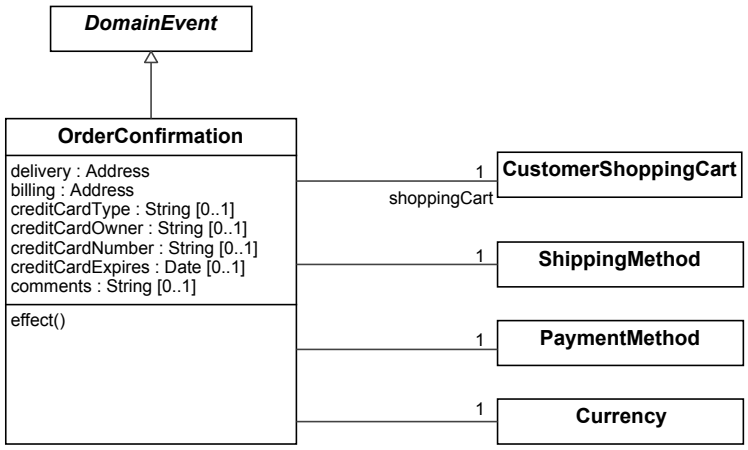
context DeleteOrderStatus::effect()
post :
if Order.allInstances.orderStatus->includes(self.orderStatus)
then self.orderStatus.status=Status::disabled
else OrderStatus.allInstances->excludes(self.orderStatus@pre)
endif

UpdateOrderStatus



context ChangeOrderStatus::effect()
post :
osc.oclIsNew() **and**
osc.oclIsTypeOf(OrderStatusChange) **and**
osc.comments = self.comments **and** osc.order = self.order **and**
osc.orderStatus = self.newOrderStatus

OrderConfirmation



```
«IniIC»
context OrderConfirmation::ShippingMethodIsEnabled(): Boolean
  body : self.shippingMethod.status= Status::enabled

«IniIC»
context OrderConfirmation::PaymentMethodIsEnabled(): Boolean
  body : self.paymentMethod.status= Status::enabled

«IniIC»
context OrderConfirmation::CurrencyIsEnabled(): Boolean
  body : self.currency.status = Status::enabled

«IniIC»
context OrderConfirmation::CreditCardDetailsNeeded(): Boolean
  body :
     self.paymentMethod.oclIsTypeOf(AuthorizeNet) or
     self.paymentMethod.oclIsTypeOf(CreditCard) or
     self.paymentMethod.oclIsTypeOf(IPayment) or
     self.paymentMethod.oclIsTypeOf(TwoCheckOut) or
     self.paymentMethod.oclIsTypeOf(PSiGate)
     implies
     creditCardType.notEmpty() and
     creditCardOwner.notEmpty() and
     creditCardNumber.notEmpty() and
     creditCardExpires.notEmpty()

«IniIC»
context OrderConfirmation::StockAllowsOrder(): Boolean
  body :
     Stock.allowCheckout or
     not Stock.checkStockLevel or
     self.shoppingCart.shoppingCartItem.product -> forAll (p | p.quantityOnHand > 0)

context  OrderConfirmation::effect()
   post theOrderIsCreated:
      o.oclIsNew() and
      o.oclIsTypeOf(Order) and
      o.customer = self.shoppingCart@pre.customer@pre and
      o.billing = self.billing and
      o.delivery = self.delivery and
      o.shippingMethod = self.shippingMethod and
      o.paymentMethod = self.paymentMethod and
      o.currency = self.currency and
      --The initial status of the order is pending
      osc.oclIsNew() and
      osc.oclIsTypeOf(OrderStatusChange) and
      osc.comments = self.comments and
      osc.orderStatus = Store.allInstances() -> any(true).defaultStatus and
      osc.order = o 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() -> one
                 (olAtt|olAtt.orderLine = ol and
                          olAtt.attribute = iAtt))))
```

```
post theShoppingCartIsRemoved:
   not self.shoppingCart@pre.oclIsKindOf(OclAny)
post updateProductQuantities:
   let productsBought:Set(Product) =
         self.shoppingCart@pre.shoppingCartItem@pre.product@pre->asSet()
   in  productsBought -> forAll (p|
         let quantityBought:PositiveInteger =
               self.shoppingCart@pre.shoppingCartItem@pre->select
                 (sc | sc.product = p).quantity -> sum()
         in
               p.quantityOrdered = p.quantityOrdered@pre + quantityBought  and
               Stock.substractStock implies
               p.quantityOnHand = p.quantityOnHand@pre – quantityBought)
```

Example test programs

```
testprogram SessionsManagement{
        co:= new Country;
        a:= new Address(country:=co);
        c:= new Customer(address:=a, primary:=a);
        //Language l has no default currency
        l:= new Language(name:='Language1', code:='L1');
        cu:=new Currency(title:='Currency1',code:='C1');
        cu2:=new Currency(title:='Currency2',code:='C2');
        //Language l2 has a default currency
        l2:=new Language(name:='Language2', code:='L2',defaultCurrency:=cu2);
        //Language l3 has no default currency
        l3:= new Language(name:='Language3', code:='L3');

        test OpenSession{
                new NewSession(currentLanguage:=l, currentCurrency:=cu) occurs;
        }

        test InvalidLogIn{
                ns:=new NewSession(currentLanguage:=l, currentCurrency:=cu) occurs;
                new LogIn(session:=ns.createdSession, customer:=c) occurs;
                //A logged-in customer cannot log in
                new LogIn(session:=ns.createdSession, customer:=c) may not occur;
                //...even if the customer tries to log in another session
                delete ns;
                ns2:=new NewSession(currentLanguage:=l, currentCurrency:=cu) occurs;
                new LogIn(session:=ns.createdSession, customer:=c) may not occur;
        }

        test InvalidLogOut{
                //We cannot log out if the customer is not logged in the session
                ns:=new NewSession(currentLanguage:=l, currentCurrency:=cu) occurs;
                new LogOut(session:=ns.createdSession, customer:=c) may not occur;
        }


        test LogInLogOutWithoutPreviousShoppingCart{
                ns:=new NewSession(currentLanguage:=l, currentCurrency:=cu) occurs;
                new LogIn(session:=ns.createdSession, customer:=c) occurs;
                new LogOut(session:=ns.createdSession, customer:=c) occurs;
        }

        test LogInLogOutWithPreviousShoppingCart{
                //The customer navigates in the store in an anonymous session
                ns:=new NewSession(currentLanguage:=l, currentCurrency:=cu) occurs;
                p:= new Product;
                assert true ns.createdSession.customer.isUndefined();

                new AddProductToShoppingCart(session:=ns.createdSession, product:=p,
                                             quantity:=1) occurs;
```

```
                assert true ns.createdSession.shoppingCart.oclIsTypeOf(AnonymousShoppingCart);
                assert equals ns.createdSession.shoppingCart.shoppingCartItem.product->asSet()
                              Set{p};

                //The customer logs in
                new LogIn(session:=ns.createdSession, customer:=c) occurs;
                assert true ns.createdSession.shoppingCart.oclIsTypeOf(CustomerShoppingCart);
                assert equals
                  ns.createdSession.shoppingCart.oclAsType(CustomerShoppingCart).customer  c;
                assert equals ns.createdSession.shoppingCart.shoppingCartItem.product->asSet()
                              Set{p};

                //The customer adds another product
                p2:=new Product;
                new AddProductToShoppingCart(session:=ns.createdSession, product:=p2,
                                             quantity:=2) occurs;

                //The customer logs out
                new LogOut(session:=ns.createdSession, customer:=c) occurs;

                //If the customer logs in again,
                //the previous customer shopping cart is restored
                new LogIn(session:=ns.createdSession, customer:=c) occurs;
                assert true ns.createdSession.shoppingCart.oclIsTypeOf(CustomerShoppingCart);
                assert equals
                   ns.createdSession.shoppingCart.oclAsType(CustomerShoppingCart).customer c;
                assert equals ns.createdSession.shoppingCart.shoppingCartItem.product->asSet()
                               Set{p,p2};

                //The session is finished
                new DeleteSession(session:=ns.createdSession) occurs;
        }

        abstract test changeCurrentLanguage
                     (Boolean switch, Language newLanguage,
                      Language expectedLanguage, Currency expectedCurrency){

                //Store Initialization
                s:=new Store(name:='FashionTShirts');
                english:=new Language(name:='English', code:='EN');
                s.defaultLanguage:=english;
                dollar:=new Currency(title:='USDollar', code:='USD', status:=#enabled);
                s.defaultCurrency:=dollar;
                usa:=new Country(name:='United States', isoCode2:='US', isoCode3:='USA');
                s.country:=usa;
                cos:=new OrderStatus;
                cosl:=new OrderStatusInLanguage(language:=english,orderStatus:=cos);
                cosl.name:='cancelled';
                s.cancelledStatus:=cos;
                dos:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=english);
                dosl.name:='pending';
                s.defaultStatus:=dos;
                //Switch to default language currency initialization
                s.switchToDefaultLanguageCurrency:=switch;

                ns:=new NewSession(currentLanguage:=l, currentCurrency:=cu) occurs;
                new SetCurrentLanguage(session:=ns.createdSession,
                                       newCurrentLanguage:=newLanguage) occurs;
                assert equals ns.createdSession.currentLanguage expectedLanguage;
                assert equals ns.createdSession.currentCurrency expectedCurrency;
        }

        //We test the effect of the "switch to default language" configuration value
        test changeCurrentLanguage(switch:=false, newLanguage:=l,
                                   expectedLanguage:=l, expectedCurrency:=cu);
        test changeCurrentLanguage(switch:=true, newLanguage:=l3,
                                   expectedLanguage:=l3, expectedCurrency:=cu);
        test changeCurrentLanguage(switch:=true, newLanguage:=l2,
                                   expectedLanguage:=l2, expectedCurrency:=cu2);
}
```

```
testprogram OrderConfirmation{
                //Store initialization
                s:=new Store(name:='FashionTShirts');
                english:=new Language(name:='English', code:='EN');
                s.defaultLanguage:=english;
                dollar:=new Currency(title:='USDollar', code:='USD', status:=#enabled);
                s.defaultCurrency:=dollar;
                usa:=new Country(name:='United States', isoCode2:='US', isoCode3:='USA');
                s.country:=usa;
                cos:=new OrderStatus;
                cosl:=new OrderStatusInLanguage(language:=english,orderStatus:=cos);
                cosl.name:='cancelled';
                s.cancelledStatus:=cos;
                dos:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=english);
                dosl.name:='pending';
                s.defaultStatus:=dos;

                //Product attributes initialization
                ssize := new Option;
                extraLarge:=new Value;
                small:=new Value;
                smallSize:=new Attribute(option:=ssize, value:=small);
                extraLargeSize:=new Attribute(option:=ssize, value:=extraLarge);

                sizeName := new StringDT(string:='size');
                new HasOptionName(option:=ssize,
                                optionName:=sizeName, optionLanguage:=english);

                extraLargeName := new StringDT(string:='extraLarge');
                new HasValueName(value:=extraLarge,
                                valueName:=extraLargeName, valueLanguage:=english);

                smallName := new StringDT(string:='small');
                new HasValueName(value:=small, valueName:=smallName, valueLanguage:=english);

                stock := new Stock;
                stock.checkStockLevel:=true;
                stock.substractStock:=true;

                //Products initialization
                fashionTShirt := new Product(netPrice:=10, quantityOnHand:=50);

                smallFashionTShirt:= new ProductAttribute(product:=fashionTShirt,
                                                        attribute:=smallSize);
                smallFashionTShirt.increment:=2;
                smallFashionTShirt.sign:=#minus;

                extraLargeFashionTShirt:= new ProductAttribute(product:=fashionTShirt,
                                                                attribute:=extraLargeSize);
                extraLargeFashionTShirt.increment:=1;
                extraLargeFashionTShirt.sign:=#plus;

                //Customer session initialization and log in
                a:= new Address(country:=usa);
                c := new Customer(address:=a,primary:=a);
                ns:=new NewSession(currentLanguage:=english, currentCurrency:=dollar) occurs;
                new LogIn(session:=ns.createdSession, customer:=c) occurs;

                fixturecomponent addRegularSizedTShirts{
                        new AddProductToShoppingCart(session:=ns.createdSession,
                                                product:=fashionTShirt,quantity:=3) occurs;
                }

                fixturecomponent addSpecialSizedTShirts{
                        new AddProductToShoppingCart(session:=ns.createdSession,
                                                product:=fashionTShirt,quantity:=2,
                                                attribute:=smallSize) occurs;
                        new AddProductToShoppingCart(session:=ns.createdSession,
                                                product:=fashionTShirt,quantity:=1,
                                                attribute:=extraLargeSize) occurs;
```

```
                }
                abstract test confirmedOrderTotal (Fixture itemsAddition, Real expectedTotal){
                        load $itemsAddition;
                        sm:= new FlatRate(status:=#enabled);
                        pm:= new Nochex(status:=#enabled);
                        oc := new OrderConfirmation
                                 (shoppingCart:=ns.createdSession.shoppingCart,
                                  currency:=dollar , shippingMethod:=sm, paymentMethod:=pm)
                                  occurs;
                        assert equals oc.orderCreated.total() expectedTotal;
                }

                test confirmedOrderTotal
                    (itemsAddition:=addRegularSizedTShirts,expectedTotal:=30.0);
                test confirmedOrderTotal
                    (itemsAddition:=addSpecialSizedTShirts,expectedTotal:=27.0);
}
```

```
testprogram CreateAndEditStatus{

        english:=new Language(name:='English', code:='EN');

        test newOrderStatus{
                pendingInEnglish:=new StringDT(string:='pending');
                nos:=new NewOrderStatus;
                newHasOrderStatusName(orderStatusName:=pendingInEnglish,
                         languageOfOrder Status:=english, orderStatusNameEvent:=nos);
                nos occurs;
                //We cannot create two order status with the same name
                nos2:=new NewOrderStatus;
                new HasOrderStatusName(orderStatusName:=pendingInEnglish,
                languageOfOrderStatus:=english, orderStatusNameEvent:=nos2);
                nos2 may not occur;
        }

        test editOrderStatus{
                pendingInEnglish:=new StringDT(string:='pending');
                nos:=new NewOrderStatus;
                new HasOrderStatusName(orderStatusName:=pendingInEnglish,
                          languageOfOrderStatus:=english, orderStatusNameEvent:=this);
                nos occurs;
                cancelledInEnglish:=new StringDT(string:='cancelled');
                nos2:=new NewOrderStatus;
                new HasOrderStatusName;
                orderStatusName:=cancelledInEnglish, languageOfOrderStatus:=english,
                                orderStatusNameEvent:=nos2);
                nos2 occurs;

                //VALID EDITIONS
                deliveredInEnglish:=new StringDT(string:='delivered');
                //It is possible to edit an order status without no name changes
                eos:=new EditOrderStatus(orderStatus:=nos.createdOrderStatus);
                new HasOrderStatusName(orderStatusName:=cancelledInEnglish,
                         languageOfOrderStatus:=english, orderStatusNameEvent:=eos);
                eos occurs;
                eos2:=new EditOrderStatus(orderStatus:=nos.createdOrderStatus);
                new HasOrderStatusName(orderStatusName:=deliveredInEnglish,
                      languageOfOrderStatus:=english, orderStatusNameEvent:=eos2);
                eos2 occurs;

                //INVALID EDITIONS
                //The edition of an order status cannot cause duplicated order status
                eos3:=new EditOrderStatus(orderStatus:=nos.createdOrderStatus);
                new HasOrderStatusName(orderStatusName:=pendingInEnglish,
                   languageOfOrderStatus:=english, orderStatusNameEvent:=this);
                eos3 may not occur;
        }
}
```

```
testprogram DeleteOrderStatus{

                english:=new Language(name:='English', code:='EN');

                //We create the order statuses
                pending:=new OrderStatus;
                posl:=new OrderStatusInLanguage(orderStatus:=pending, language:=english);
                posl.name:='pending';

                cancelled:=new OrderStatus;
                cosl:=new OrderStatusInLanguage(orderStatus:=cancelled, language:=english);
                cosl.name:='cancelled';

                delivered:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=delivered, language:=english);
                dosl.name:='delivered';

                returned:=new OrderStatus;
                rosl:=new OrderStatusInLanguage(orderStatus:=returned, language:=english);
                rosl.name:='returned';

                //We initialize an store
                usa:=new Country(name:='USA', isoCode2:='US', isoCode3:='USA');
                euro:=new Currency(title:='Euro', code:='EUR', status:=#enabled);

                //Store configuration
                s:=new Store;
                s.defaultLanguage:=english;
                s.defaultCurrency:=euro;
                s.country:=usa;
                s.defaultStatus:=pending;
                s.cancelledStatus:=cancelled;
                //Stock configuration
                stock := new Stock;
                stock.checkStockLevel:=true;
                stock.substractStock:=true;
                //Products configuration
                standardLaptop := new Product(netPrice:=949, quantityOnHand:=300);
                //Payment methods configuration
                pm:=new CashOnDelivery(status:=#enabled);
                //Shipping configuration
                sm:=new PerItem(status:=#enabled, handlingFee:=5, cost:=10);

                //We create an order which, initially, has the pending status (by default)
                //Customer initialization and login
                a:= new Address(country:=usa);
                c := new Customer(address:=a,primary:=a);
                ns:=new NewSession(currentLanguage:=english, currentCurrency:=euro) occurs;
                new LogIn(session:=ns.createdSession, customer:=c) occurs;
                new AddProductToShoppingCart(session:=ns.createdSession,
                                             product:=standardLaptop,quantity:=2) occurs;
                oc := new OrderConfirmation(shoppingCart:=ns.createdSession.shoppingCart,
                          currency:=euro , shippingMethod:=sm, paymentMethod:=pm, billing:=a)
                occurs;
                orderCreated:=oc.orderCreated;

                test deleteOrderStatusIfNoOrdersUsedIt{
                        //If the order status has not been used, it can be deleted at all
                        new DeleteOrderStatus(orderStatus:=delivered) occurs;
                        assert false OrderStatus.allInstances->exists(orderStatusInLanguage
                                        ->any(language=english).name='delivered');
                }

                test deleteStoreDefaultOrderStatus{
                        //A default status of the store cannot be deleted
                        new DeleteOrderStatus(orderStatus:=pending) may not occur;
                        new DeleteOrderStatus(orderStatus:=cancelled) may not occur;
                }

                test deleteOrderStatusIfItIsTheCurrentStatusOfAnOrder{
                        //If the order status is the current status of an order, the deletion
```

```
                        //only changes its status to disabled
                        new UpdateOrderStatus(order:=orderCreated,newOrderStatus:=delivered)
                        occurs;
                        new DeleteOrderStatus(orderStatus:=delivered) may not occur;
                }

                test deleteOrderStatusIfItWasTheStatusOfAnOrder{
                        //If the order status was the status of an order (not the current
                        //status) the system disables the order status.
                        New UpdateOrderStatus(order:=orderCreated,newOrderStatus:=delivered)
                        occurs;
                        New UpdateOrderStatus(order:=orderCreated,newOrderStatus:=returned)
                        occurs;
                        new DeleteOrderStatus(orderStatus:=delivered) occurs;
                        assert equals delivered.status #disabled;
                }
}
```

Finally, we present a test program that test a typical scenario of the use case “Place and Order” which is the main functionallity of the system from the customers point of view.

```
testprogram PlaceAndOrder{

                //STORE INITIALIZATION
                //Location, currencies and languages
                spain:=new Country(name:='Spain', isoCode2:='ES', isoCode3:='ESP');
                catalonia:=new Zone(name:='Catalonia', code:='CAT', country:=spain);
                english:=new Language(name:='English', code:='EN');
                euro:=new Currency(title:='Euro', code:='EUR', status:=#enabled);

                //Store configuration
                s:=new Store(name:='CustomizedComputers');
                s.defaultLanguage:=english;
                s.defaultCurrency:=euro;
                s.country:=spain;
                s.zone:=catalonia;

                //Default order status
                cancelled:=new OrderStatus;
                cosl:=new OrderStatusInLanguage(language:=english,orderStatus:=cancelled);
                cosl.name:='cancelled';
                s.cancelledStatus:=cancelled;
                pending:=new OrderStatus;
                dosl:=new OrderStatusInLanguage(orderStatus:=pending, language:=english);
                dosl.name:='pending';
                s.defaultStatus:=pending;
                delivered:=new OrderStatus;
                deosl:=new OrderStatusInLanguage(orderStatus:=delivered, language:=english);
                deosl.name:='delivered';

                //Stock configuration
                stock := new Stock;
                stock.checkStockLevel:=true;
                stock.substractStock:=true;

                //Product attributes initialization
                warranty := new Option;
                premium:=new Value;
                plus:=new Value;

                premiumWarranty:=new Attribute(option:=warranty, value:=premium);
                plusWarranty:=new Attribute(option:=warranty, value:=plus);

                warrantyName := new StringDT(string:='Warranty');
                new HasOptionName(option:=warranty,
                                optionName:=warrantyName, optionLanguage:=english);
```

```
premiumName := new StringDT(string:='Premium');
new HasValueName(value:=premium,
                 valueName:=premiumName, valueLanguage:=english);

plusName := new StringDT(string:='Plus');
new HasValueName(value:=plus, valueName:=plusName, valueLanguage:=english);

//Products initialization
standardLaptop := new Product(netPrice:=949, quantityOnHand:=300);

plusWarrantyLaptop:= new ProductAttribute(product:=standardLaptop,
                                          attribute:=plusWarranty);
plusWarrantyLaptop.increment:=60;
plusWarrantyLaptop.sign:=#plus;

premiumWarrantyLaptop:= new ProductAttribute(product:=standardLaptop,
                                             attribute:=premiumWarranty);
premiumWarrantyLaptop.increment:=112;
premiumWarrantyLaptop.sign:=#plus;

illustratedStartGuide:= new Product(netPrice:=15,quantityOnHand:=50);

//Taxes configuration
spanishVAT:=new TaxZone(name:='SpanishVAT');
spanishVAT.zone:=catalonia;

//We allow two types of VAT: general VAT (16%) and super-reduced VAT(4%)
general:=new TaxClass(name:='generalVAT');
superreduced:=new TaxClass(name:='super-reducedVAT');

//For each TaxClass, there is a different tax rate applied in each zone
generalRate:=new TaxRate(taxClass:=general, taxZone:=spanishVAT);
generalRate.rate:=16;
generalRate.priority:=1;

superReducedRate:=new TaxRate(taxClass:=superreduced, taxZone:=spanishVAT);
superReducedRate.rate:=4;
superReducedRate.priority:=1;

standardLaptop.taxClass:=general;
illustratedStartGuide.taxClass:=superreduced;

//Payment methods configuration
pm:=new CashOnDelivery(status:=#enabled);

//Shipping configuration
sm:=new PerItem(status:=#enabled, handlingFee:=5, cost:=10);

test placeAndOrder{
        //Customer initialization
        a:= new Address(country:=spain, zone:=catalonia, state:='Catalonia');
        c := new Customer(address:=a,primary:=a);
        //The customer logs in
        ns:=new NewSession(currentLanguage:=english, currentCurrency:=euro)
       occurs;

        /*
        The customer adds to the shopping cart the following items:
                - 2 standard laptops with no warranty
                - Standard laptop with Premium warranty
                - Illustrated Start guide
        */

        new AddProductToShoppingCart(session:=ns.createdSession,
                                     product:=standardLaptop,quantity:=2) occurs;
        new AddProductToShoppingCart(session:=ns.createdSession,
                                        product:=standardLaptop,quantity:=1,
                                        attribute:=premiumWarranty) occurs;
        new AddProductToShoppingCart(session:=ns.createdSession,
                           product:=illustratedStartGuide,quantity:=1) occurs;
```

```
                    new LogIn(session:=ns.createdSession, customer:=c) occurs;

                    sc:=ns.createdSession.shoppingCart;
                    oc := new OrderConfirmation
                          (shoppingCart:=ns.createdSession.shoppingCart, currency:=euro,
                           shippingMethod:=sm, paymentMethod:=pm, billing:=a) occurs;
                    orderCreated:=oc.orderCreated;

                    assert equals orderCreated.orderLine.product->asSet()->size() 2;
                    assert equals orderCreated.orderLine
                                  ->select(product=standardLaptop).quantity->sum() 3;
                    assert equals orderCreated.orderLine
                                ->select(product=illustratedStartGuide).quantity->sum() 1;

                    assert equals standardLaptop.quantityOnHand 297;
                    assert equals illustratedStartGuide.quantityOnHand 49;

                    /*
                    Order total details
                    ========================
                    2 x standard laptop (no warranty)       x 949   =       1898,00
                    1 x standard laptop (premium warranty)  x 1061  =       1061,00
                    Subtotal ..........................................     2959,00
                    VAT 16%............................................      473,44
                    Total (16%)........................................     3432,44

                    1 x illustrated start guide             x 15    =         15,00
                    Subtotal ..........................................       15,00
                    VAT 4%.............................................        0,60
                    Total (4%).........................................       15,60

                    ---Shipping costs (Per Item)
                    Handling fee ......................................       5,00
                    4 x Per Item Rate                       x 10    =         40,00

                    Order Total _______________________________________     3493,04
                    */

                    assert equals orderCreated.total() 3493.04;

                    //The store administrator can change the status of the order...
                    new UpdateOrderStatus(order:=orderCreated,newOrderStatus:=delivered)
                    occurs;
                    assert equals orderCreated.orderStatus Sequence{pending,delivered};

                    //...or he can cancel the order (order information cannot be deleted)
                    new CancelOrder(order:=orderCreated) occurs;
                    assert equals
                          orderCreated.orderStatus  Sequence{pending,delivered,cancelled};
              }
}
```

10. CONCLUSIONS

Conceptual schemas can be tested

We have seen that, **like any software artifact, conceptual schemas of information systems can be tested** with the goal of "uncover faults by triggering failures" [23]. We have shown that testing conceptual schemas has some similarities with testing programs, but there are important differences.

A catalog of test kinds applicable to conceptual schemas

We have presented a list of six kinds of tests that can be applied to conceptual schemas. Some of these test kinds require conceptual schemas that include all structural and behavioral aspects, but we have seen that it makes sense to test also incomplete conceptual schemas. Small fragments consisting of a few entity and relationship types, integrity constraints and derivation rules can be tested to uncover their faults and, therefore, to increase their quality [20].

A language for writing tests of conceptual schemas

We have presented CSTL, a textual procedural language for writing automated tests of executable conceptual schemas written in UML/OCL. The main features of the language have been illustrated by examples taken from its application to the osCommerce case study. As far as we know, this is the first proposal of a language for testing conceptual schemas designed in the style of the modern xUnit testing frameworks. Tests written in CSTL may be automatically executed as many times as needed.

A test processor for executing CSTL programs

We have implemented a Test Processor that manages and executes CSTL programs. It includes a test interpreter that coordinates the execution of the tests and invokes the services of an information processor, which we have implemented reusing USE [14].

CSTL application in a case study

We have applied our proposal to the conceptual schema of a real e-commerce system. The experiences acquired by applying the CSTL language to a real case constitute a base to study and propose future improvements of our proposal.

By applying tests to the osCommerce conceptual schema we also found errors in the conceptual schema, some of them difficult to be detected without testing and executing the model.

New directions for research in conceptual modeling

We believe that our work opens new directions for research and development in conceptual modeling:

- It is necessary to develop **a methodology for testing conceptual schemas**. Here we have focused on the testing language and the test processor but we need to know how to use them in professional projects in order to get the maximum benefit. In particular, it seems interesting to develop a test-driven conceptual modeling methodology, similar to the popular Test-Driven Development [4].

- Similar to program code coverage, it is necessary to develop **coverage criteria that measure the degree to which a conceptual schema has been tested**. Such criteria are useful to determine the parts of a conceptual schema that need more tests.

- Conceptual schema testing should be **integrated with other verification techniques**, and the test processor should be integrated **with the other tools** of a comprehensive development environment [5].

11. REFERENCES

1. Andrea, J. Envisioning the Next Generation of Functional Testing Tools. *IEEE SOFTWARE*, (2007), 58-66.

2. Association for Information Systems. AIS: Design Research in Information Systems. http://home.aisnet.org/displaycommon.cfm?an=1&subarticlenbr=279.

3. Baker, P., Dai, Z.R., Grabowski, J., Schieferdecker, I., Haugen, O., and Williams, C. *Model-Driven Testing: Using the UML Testing Profile*. Springer, 2007.

4. Beck, K. *Test-driven Development: By Example*. Addison-Wesley Professional, 2003.

5. Bouzeghoub, M., Kedad, Z., and Métais, E. . CASE Tools: Computer Support for Conceptual Modeling. In *Advanced Database Technology and Design*. Artech House, 2000, 439-483.

6. Bremen University. USE: A UML based Specification Environment. http://www.db.informatik.uni-bremen.de/projects/USE/use-documentation.pdf.

7. Briand, L. and Labiche, Y. A UML-Based Approach to System Testing. *Software and Systems Modeling 1*, 1 (2002), 10-42.

8. Cabot, J. and Teniente, E. Constraint Support in MDA Tools: A Survey. *LECTURE NOTES IN COMPUTER SCIENCE 4066*, (2006), 256.

9. CARE Technologies. OLIVANOVA. http://www.care-t.com/index.asp.

10. Dinh-Trong, T., France, R.B., Hamilton, M., and Wilkins, B. UMLAnT: An Eclipse Plugin for Animating and Testing UML Designs. *Proc. Eclipse Technology eXchange Workshop, Conf. Object-Oriented Programming, Systems, Languages and Applications*.

11. Dresden University. Dresden OCL. http://dresden-ocl.sourceforge.net/index.php.

12. Gamma, E. and Beck, K. JUnit: A cook's tour. *Java Report 4*, 5 (1999), 27-38.

13. Glass, R.L. An Ancient (but Still Valid?) Look at the Classification of Testing. *Software, IEEE 25*, 6 (2008), 112.

14. Gogolla, M., Bohling, J., and Richters, M. Validating UML and OCL models in USE by automatic snapshot generation. *Software and Systems Modeling 4*, 4 (2005), 386-398.

15. van Griethuysen, J.J. and TC97, I. Concepts and Terminology for the Conceptual Schema and the Information Base. (1982).

16. Insfrán, E., Pelechano, V., and Pastor, O. Conceptual Modeling in the eXtreme. *Information and Software Technology 44*, 11 (2002), 659-669.

17. Insfrán, E., Pelechano, V., and Pastor, O. Conceptual Modeling in the eXtreme. *Information and Software Technology 44*, 11 (2002), 659-669.

18. Janzen, D. and Saiedian, H. Test-Driven Development: Concepts, Taxonomy, and Future Direction. *COMPUTER*, (2005), 43-50.

19. Larman, C. *Applying UML and patterns*. Prentice Hall, 2005.

20. Lindland, O.I., Sindre, G., and Solvberg, A. Understanding quality in conceptual modeling. *Software, IEEE 11*, 2 (1994), 42-49.

21. Martin, R.C., Melnik, G., and Inc, O.M. Tests and Requirements, Requirements and Tests: A Möbius Strip. *Software, IEEE 25*, 1 (2008), 54-59.

22. Mellor, S.J. and Balcer, M.J. *Executable UML: A Foundation for Model-Driven Architecture*. Addison-Wesley Professional, 2002.

23. Meyer, B. Seven Principles of Software Testing. *Computer 41*, 8 (2008), 99-101.

24. NETO, P., RESENDE, R., and PADUA, C. A Method for Information Systems Testing Automation. *Lecture Notes in Computer Science*, (2005), 3520.504-518.

25. Nijssen, G.M. and Halpin, T.A. *Conceptual schema and relational database design: a fact oriented approach*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1989.

26. Olivé, A. Derivation Rules in Object-Oriented Conceptual Modeling Languages. *CAiSE '03, LNCS 2681*, (2003), 404-420.

27. Olivé, A. Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. *CAiSE*, Springer (2005), 1-15.

28. Olivé, A. and Raventós, R. Modeling events as entities in object-oriented conceptual modeling languages. *Data & Knowledge Engineering 58*, 3 (2006), 243-262.

29. Olivé, A. *Conceptual Modeling of Information Systems*. Springer, 2007.

30. OMG. *MDA Guide Version 1.0.1*. 2003.

31. OMG. *UML Testing Profile. Version 1.0*. 2005.

32. OMG. *Object Constraint Language. Version 2.0*. 2006.

33. OMG. *UML Supraestructure version 2.1.2*. 2007.

34. Ostroff, J.S., Paige, R.F., Makalsky, D., and Brooke, P.J. E-Tester: a Contract-Aware and Agent-Based Unit Testing Framework for Eiffel. *JOURNAL OF OBJECT TECHNOLOGY 4*, 7.

35. Ostroff, J.S. and Torshizi, F.A. Testable Requirements and Specifications. *LECTURE NOTES IN COMPUTER SCIENCE 4454*, (2007), 17.

36. Pastor, O. and Molina, J.C. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer Verlag, 2007.

37. Pilskalns, O., Andrews, A., Knight, A., Ghosh, S., and France, R. Testing UML designs. *Information and Software Technology 49*, 8 (2007), 892-912.

38. Richters, M. and Gogolla, M. Validating UML Models and OCL Constraints. *LNCS*, (2000), 265-277.

39. Tassey, G. The economic impacts of inadequate infrastructure for software testing, final report. *National Institute of Standards and Technology*, (2002).

40. Tort, A. Test-Driven Conceptual Modeling: A Method and a Tool. *PhD Workshop, ER 2008*, .

41. Tort, A. The osCommerce Conceptual Schema. 2007. http://guifre.lsi.upc.edu/Sudoku.PDF.

42. Vos, T.E.J. and Sánchez, S. Especificar casos de testeo tan fácilmente como hacer una tortilla española. *Novática*, 191 (2008), 45.

43. Vrandecic, D. and Gangemi, A. Unit Tests for Ontologies. *LECTURE NOTES IN COMPUTER SCIENCE 4278*, (2006), 1012.

44. Zhang, Y., Inc, M., and Arlington Heights, I.L. Test-driven modeling for model-driven development. *Software, IEEE 21*, 5 (2004), 80-86.