CSTL: A Conceptual Schema Testing Language

Albert Tort Pugibet

Advisor: Antoni Olivé Ramon
<table>
<thead>
<tr>
<th><strong>Title of thesis</strong></th>
<th>CSTL: A Conceptual Schema Testing Language</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author</strong></td>
<td>Albert Tort Pugibet</td>
</tr>
<tr>
<td><strong>Advisor</strong></td>
<td>Antoni Olivé Ramon</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td>January 15th, 2008</td>
</tr>
</tbody>
</table>
| **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 colleagues 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** ............................................................................................................... 1
   1.1. Master Thesis Purpose ............................................................................................ 2
   1.2. Document Structure .............................................................................................. 2

2. **THE RESEARCH APPROACH** .................................................................................... 3

3. **BASIC CONCEPTS AND NOTATION** ......................................................................... 5
   3.1. The Conceptual Modeling Activity ........................................................................... 5
   3.2. The Conceptual Schema Under Test ....................................................................... 5
   3.3. The Structural Schema .......................................................................................... 5
   3.4. The Behavioral Schema ......................................................................................... 7

4. **TESTING CONCEPTUAL SCHEMAS** ......................................................................... 8
   4.1. Schema Testing vs. Software Testing ...................................................................... 8
   4.2. Functional Requirements and Tests ........................................................................ 9

5. **RELATED WORK** ......................................................................................................... 10
   5.1. CASE tools .......................................................................................................... 10
   5.2. The challenge of executing conceptual schemas .................................................. 10
   5.3. The testing activity ............................................................................................... 11
   5.4. Testing models ..................................................................................................... 12

6. **TEST KINDS APPLICABLE TO CONCEPTUAL SCHEMAS** ................................. 13
   6.1. Information Base State ......................................................................................... 13
   6.2. Check that a given IB state is consistent ............................................................ 15
   6.3. Check that a given IB state is inconsistent .......................................................... 16
   6.4. Check the contents of a given IB state ................................................................. 17
6.5. Check that an event or query may not occur .......................................................... 17
6.6. Check that a domain event occurs ........................................................................ 18
6.7. Check that a predefined query produces the expected results ............................ 19

7. CSTL SPECIFICATION .................................................................................................. 20

7.1. Five Design Principles of CSTL ............................................................................. 20
7.2. Test Program Structure ............................................................................................ 21
7.3. Kinds of Test Cases .................................................................................................. 22
7.4. CSTL Types and Value Expressions ....................................................................... 23
7.5. Test Verdicts ............................................................................................................ 24
7.6. Language syntax ...................................................................................................... 25
7.7. CSTL statements ...................................................................................................... 27
    7.7.1. State statements ............................................................................................... 27
    7.7.2. Variable statements ......................................................................................... 30
    7.7.3. Assert statements ............................................................................................ 31
    7.7.4. Control flow statements .................................................................................. 33

8. THE CSTL ENVIRONMENT ......................................................................................... 36

9. APPLICATION TO THE OCOMMERCE CONCEPTUAL SCHEMA .................... 39

9.1. Main domain concepts ............................................................................................. 40
9.2. Store Data ................................................................................................................ 42
9.3. Configuration values ............................................................................................... 46
9.4. Payment methods .................................................................................................... 54
9.5. Shipping methods .................................................................................................... 59
9.6. Languages ................................................................................................................ 64
9.7. Currencies ............................................................................................................... 69
9.8. Location & Taxes .................................................................................................... 76
9.9. Products .................................................................................................................... 95
9.10. Product attributes and options ............................................................................. 104
9.11. Product categories ............................................................................................... 124
9.12. Specials .................................................................................................................. 133
9.13. Manufacturers ....................................................................................................... 137
9.15. Newsletters ............................................................................................................ 150
9.16. Customers............................................................................................................. 158
9.17. Reviews ............................................................................................................. 180
9.18. Shopping carts & Orders................................................................................... 185

10. CONCLUSIONS ...................................................................................................... 217

11. REFERENCES ....................................................................................................... 219

APPENDIX A : osCommerce Conceptual Schema in the USE format..................... 222

APPENDIX B : osCommerce methods................................................................. 305
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 thesis’ website (www.lsi.upc.edu/~atort/cstl): The complete master thesis report; the appendix 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.

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].
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 $\text{effect}()$ 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 $\text{effect}()$ 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 $\text{effect}()$ 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 $\text{effect}()$ 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 those 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 master 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 master 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.
Fig. 3. An example instantiation of Figure 2.

In a test case, we define that \textit{entityID} is a new instance of the entity types \textit{EntityType}_1,...,\textit{EntityType}_n with the statement:

\begin{verbatim}
entityID := new EntityType1,..., EntityTypen;
\end{verbatim}

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

\begin{verbatim}
entityID.att := val;
\end{verbatim}

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

\begin{verbatim}
entityID.role := participants;
\end{verbatim}

Instances of an n-ary UML association \textit{Assoc} with roles \textit{r}_1,...,\textit{r}_n are created with the statement:

\begin{verbatim}
new Assoc(r_1:= entityID, ..., r_n:= entityID);
\end{verbatim}

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 \textit{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 sci2, but not for sci1.

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:

```cstl
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 $sc_1.price$ is missing. The test can Pass if the derivation rule of $ShoppingCart &: price$ is corrected as indicated in Figure 2, and the following statement is added to Figure 3:

$$sc_1.price := 15$$

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:

```nlu
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 $sc_1.attribute$ in Figure 3 to:

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

Now the test of $check inconsistency$ will Pass because $sci1$ 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:

\[ \text{assert true booleanExpression;} \]

where \(\text{booleanExpression}\) is an OCL expression over the types of the IB and the variables of the test case. The verdict is Pass if \(\text{booleanExpression}\) 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: \text{assert false}, \text{assert equals} and \text{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:

\[ \text{assert false sci1.price = sci2.price;} \]

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:

\[ \text{new [DomainEventType|Query] (c_1:= expression_1,..., c_n:= expression_n) may not occur;} \]
where expression, 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 pa1 (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, 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:

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

Then, the test:

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

gives the verdict Pass if the method of the operation correctly deletes `pa3`.

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:

```plaintext
queryID := new Query (c_1:=expression_1,..., c_n:=expression_n)    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.
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.
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, defines and assigns 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.

Fig. 4. CSTL metamodel fragment of test programs.
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]. Moreover, 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.
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.

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.

```plaintext
<grammar>
  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.

```cstl

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 c₁ with the value valueExpression₁,... and the characteristic cₙ with the value valueExpressionₙ) 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 entity 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;
```
“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;
```

“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;
```

“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**

```plaintext
assert false booleanExpression;
```

**Pattern Sentence**

“Assert that the expression `booleanExpression` is false in the current state of the IB”.

### Assert equals

**Syntax**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
if booleanExpression\_i then statements\_i
[else if booleanExpression\_i+1 then statements\_i+1]
...
[else if booleanExpression\_n-1 then statements\_n-1]
[else statements\_n]
endif
```

**Pattern Sentence**

“If the expression booleanExpression\_i evaluates true the set of statements statements\_i is executed. Otherwise, the set of statements statements\_n 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 `valueExpr1` the set of statements `statements` are repeated until `varID` is equal to the value of `valueExpr2`. In each iteration the value obtained by evaluating the expression `valueExpr3` 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 (`valueExpr1`).

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**

```plaintext
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.

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.

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 \texttt{allInstances} expressions does not admit parenthesis like in the standard OCL.

\begin{verbatim}
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;
    smallFashionTShirt := new ProductAttribute(product := fashionTShirt, attribute := smallSize);
    extraLargeFashionTShirt := new ProductAttribute(product := fashionTShirt, attribute := extraLargeSize);

    //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.quantity := 3;
        item1.shoppingCart := sc;
    }

    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);
}
\end{verbatim}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig10.png}
\caption{CSTL program for testing order confirmation}
\end{figure}
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 preceded 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:

```java
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:

```java
context OrderConfirmation inv _iniIC_ShippingMethodIsEnabled:
    self.shippingMethod.status= #enabled
```
9. APPLICATION TO THE OSMOCOMMERCE 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 which 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.

![Diagram of Store schema]

**[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**

```java
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:

   - NameChange
   - OwnerChange
   - EMailAddressChange
   - EMailFromChange
   - ExpectedSortOrderChange
   - ExpectedSortFieldChange
   - SendExtraOrderChange
   - DisplayCartAfterAddingProductChange
   - AllowGuestToTellAFriendChange
   - DefaultSearchOperatorChange
   - StoreAddressAndPhoneChange
   - TaxDecimalPlacesChange
   - DisplayPricesWithTaxChange
   - SwitchToDefaultLanguageCurrencyChange
   - 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

```plaintext
domainName = 'NameChange'
context NameChange::effect()
post: self.myStore.name = self.newName
```

CountryChange

```plaintext
domainName = 'CountryChange'
context CountryChange::effect()
post: myStore.country = self.newCountry
```

Example test program

```plaintext
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
  s.country := spain;
  cos := new OrderStatus;
  cos1 := new OrderStatusInLanguage(language:='EN', orderStatus:='cancelled');
  cos1.name := 'cancelled';
}```
9.3. Configuration values

Structural schema

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

<table>
<thead>
<tr>
<th>MinimumValues</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName : PositiveInteger</td>
</tr>
<tr>
<td>lastName : PositiveInteger</td>
</tr>
<tr>
<td>dateOfBirth : PositiveInteger</td>
</tr>
<tr>
<td>eMailAddress : PositiveInteger</td>
</tr>
<tr>
<td>streetAddress : PositiveInteger</td>
</tr>
<tr>
<td>companyName : Natural</td>
</tr>
<tr>
<td>postCode : PositiveInteger</td>
</tr>
<tr>
<td>city : PositiveInteger</td>
</tr>
<tr>
<td>state : PositiveInteger</td>
</tr>
<tr>
<td>telephoneNumber : PositiveInteger</td>
</tr>
<tr>
<td>password : PositiveInteger</td>
</tr>
<tr>
<td>creditCardOwnerName : PositiveInteger</td>
</tr>
<tr>
<td>creditCardNumber : PositiveInteger</td>
</tr>
<tr>
<td>reviewText : Natural</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MaximumValues</th>
</tr>
</thead>
<tbody>
<tr>
<td>addressBookEntries : Natural</td>
</tr>
</tbody>
</table>

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.

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

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

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

```
context CustomerDetails::enableDownload: Boolean
daysExpiryDelay : Natural
maximumNumberOfDownloads : Natural
```

The system allows configuring some options about the stock administration.

```
context Stock::allowCheckout : Boolean
stockReOrderLevel : Natural
```
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
   → LastName MinimumChange
   → 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:
   
   $\Rightarrow$ 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:
   
   $\Rightarrow$ GenderCustomerDetailChange  
   $\Rightarrow$ DateOfBirthCustomerDetailChange  
   $\Rightarrow$ CompanyCustomerDetailChange  
   $\Rightarrow$ SuburbCustomerDetailChange  
   $\Rightarrow$ 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:
   - \[→\text{CheckLevelStockConfigurationChange}\]
   - \[→\text{SubtractStockConfigurationChange}\]
   - \[→\text{AllowCheckoutStockConfigurationChange}\]
   - \[→\text{ReorderLevelStockConfigurationChange}\]
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**

```
DomainEvent

PasswordMinimumChange

newMinimum : PositiveInteger

\text{context} \ \text{PasswordMinimumChange::effect()}

\text{post} : \ MinimumValues.password = self.newMinimum
```

**CreditCardNumberMinimumChange**

```
DomainEvent

CreditCardNumberMinimumChange

newMinimum : PositiveInteger

\text{context} \ \text{CreditCardNumberMinimumChange::effect()}

\text{post} : \ MinimumValues.creditCardNumber = self.newMinimum
```
AddressBookEntriesMaximumChange

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

GenderCustomerDetailChange

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

MaximumPackageWeightShippingConfigurationChange

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

<InInIC>
TypicalPackageTareWeightShippingConfigurationChange

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

```plaintext
context TypicalPackageTareWeightShippingConfigurationChange::valueDoesNotExceedMaxWeight():Boolean
body : self.newValue < ShippingAndPackaging.maximumPackageWeight
```

MaximumNumberDownloadConfigurationChange

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

CheckLevelStockConfigurationChange

```plaintext
context CheckLevelStockConfigurationChange::effect()
post : Stock.checkStockLevel= self.newValue
```
Example test program

```haskell
example ConfigurationValues[
    // We create an instance of the entity types
    // MaximumValues and MinimumValues (multiple classification)
    configurationValues := new MaximumValues, MinimumValues,
                          CustomerDetails, ShippingAndPackaging, Download, Stock;
    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 MaximumNumberOfDownloadConfigurationChange(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.
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:
   
   ![Installation methods](image)

   1. InstallAuthorizeNetPaymentMethod
   2. InstallCreditCardPaymentMethod
   3. InstallCashOnDeliveryPaymentMethod
   4. InstallIPaymentPaymentMethod
   5. InstallCheckMoneyPaymentMethod
   6. InstallNochexPaymentMethod
   7. InstallPayPalPaymentMethod
   8. InstallTwoCheckOutPaymentMethod
   9. InstallPSiGatePaymentMethod
   10. 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]
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**

```plaintext
context InstallCreditCardPaymentMethod::paymentMethodIsNotInstalled():Boolean
body : CreditCard.allInstances() -> isEmpty()
context InstallCreditCardPaymentMethod::effect()
post : pm.oclIsNew() and pm.oclIsTypeOf(CreditCard) and pm.status=Status::enabled
```

**UninstallCreditCardPaymentMethod**

```plaintext
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**

```
context  EditCreditCardPaymentMethod::paymentMethodIsInstalled():Boolean
body :  CreditCard.allInstances() -> notEmpty()
context  EditCreditCardPaymentMethod::atLeastOneEnabled():Boolean
body :  
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;
}
```
9.5. Shipping methods

Structural schema

The system allows operating with different shipping methods.

 CONTEXT ShippingMethod::atLeastOneEnabled: Boolean  
 BODY: ShippingMethod.allInstances() -> select (sm | sm.status=Status::enabled) -> size() >= 1

[IC1] There is at least one enabled shipping method.
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.
   - \[\text{InstallZoneRatesShippingMethod}\]
   - \[\text{InstallFlatRateShippingMethod}\]
   - \[\text{InstallPerItemShippingMethod}\]
   - \[\text{InstallTableRateShippingMethod}\]
   - \[\text{InstallUSPostalServiceShippingMethod}\]
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.
   - \[\text{UninstallZoneRatesShippingMethod}\]
   - \[\text{UninstallFlatRateShippingMethod}\]
   - \[\text{UninstallPerItemShippingMethod}\]
   - \[\text{UninstallTableRateShippingMethod}\]
   - \[\text{UninstallUSPostalServiceShippingMethod}\]
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**

- **DomainEvent**

  - **InstallPerItemShippingMethod**

    - **effect()**

**«InitC»**

- **context** InstallPerItemShippingMethod::ShippingMethodIsNotInstalled():Boolean
  - **body**: PerItem.allInstances() -> isEmpty()

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

**UninstallPerItemShippingMethod**

- **DomainEvent**

  - **UninstallPerItemShippingMethod**

    - **effect()**

**«InitC»**

- **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)->oclIsKindOf(OclAny)
EditPerItemShippingMethod

context EditPerItemShippingMethod::paymentMethodsInstalled():Boolean
body: PerItem.allInstances() -> notEmpty()

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;
  }
}
9.6. Languages

Structural schema

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

<table>
<thead>
<tr>
<th>Language</th>
<th>Currency</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : String</td>
<td>defaultCurrency</td>
</tr>
<tr>
<td>code : String</td>
<td></td>
</tr>
<tr>
<td>image : File [0..1]</td>
<td></td>
</tr>
<tr>
<td>directory : String</td>
<td></td>
</tr>
<tr>
<td>sortOrder : Natural</td>
<td></td>
</tr>
</tbody>
</table>

[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:
   \[\Rightarrow \text{NewLanguage}\]
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:
   \[\Rightarrow \text{EditLanguage}\]
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:
   \[\Rightarrow \text{DeleteLanguage}\]
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:
      \[\Rightarrow \text{SetDefaultLanguage}\]
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:
      \[\Rightarrow \text{SetCurrentLanguage}\]
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:
   
   $\rightarrow SetDefaultLanguage$

Events

NewLanguage

```
DomainEvent

NewLanguage

name : String
code : String
effect()

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

context NewLanguage::effect()
post :
IlocsIsNew() and
I.oclIsTypeOf(Language) and
I.name = self.name and
I.code = self.code and
I.defaultCurrency = self.defaultCurrency
```
### EditLanguage

**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

**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:=catalan) occurs;
    }
}
```
9.7. Currencies

Structural schema

**osCommerce** allows working with different currencies.

<table>
<thead>
<tr>
<th>Currency</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>title : String</td>
<td>code : String</td>
</tr>
<tr>
<td>symbolLeft : String [0..1]</td>
<td>symbolRight : String [0..1]</td>
</tr>
<tr>
<td>decimalPlaces : Natural value : Decimal</td>
<td>lastUpdate : DateTime [0..1]</td>
</tr>
<tr>
<td>status : Status</td>
<td></td>
</tr>
</tbody>
</table>

<<enumeration>>

<table>
<thead>
<tr>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
</tr>
<tr>
<td>disabled</td>
</tr>
</tbody>
</table>

[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)

At least one currency is enabled

**context** Currency::codeAndTitleAreUnique : Boolean
**body** : Currency.allInstances() ->one(status=Status::enabled)
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:
   
   $\rightarrow$ 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:
   
   $\rightarrow$ 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:
   \[ \rightarrow \text{DeleteCurrency} \]
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:
      \[ \rightarrow \text{SetDefaultCurrency} \]

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:
      \[ \rightarrow \text{SetCurrentCurrency} \]

2c. The currency is the currency of an order:
   2c1. The system changes the status of the currency to disable.
      \[ \rightarrow \text{CurrencyStatusChange} \]
   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:
   \[ \rightarrow \text{UpdateCurrencyValueChange} \]

**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:
   \[ \rightarrow \text{SetDefaultCurrency} \]
Events

NewCurrency

DomainEvent

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

DomainEvent

ExistingCurrencyEvent

Currency

EditCurrency

newTitle : String
newCode : String
newSymbolLeft : String [0..1]
newSymbolRight : String [0..1]
newDecimalPlaces : Natural
newValue : Decimal

effect()
**-InitI-**

**context** EditCurrency::currencyDoesNotExist(): Boolean

**body:**

\[\neg ((\text{Currency.allInstances}-\text{Set}\{\text{self.currency}\})\rightarrow\text{exists}(\text{title}=\text{self.newTitle}\text{ or code}=\text{self.newCode}))\]

**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**

\[
\begin{array}{c}
\text{Currency} \\
1 \\
\text{ExistingCurrencyEvent} & \text{DomainEvent} \\
\text{DeleteCurrency} & \\
\text{effect()}
\end{array}
\]

**-InitI-**

**context** DeleteCurrency::AtLeastTwoCurrencies(): Boolean

**body:** Currency.allInstances() \rightarrow\text{size()} \geq 2

**context** DeleteCurrency::effect()

**post:** \neg self.currency@pre.oclIsKindOf(OclAny)

**SetDefaultCurrency**

\[
\begin{array}{c}
\text{Currency} \\
1 \\
\text{ExistingCurrencyEvent} & \text{DomainEvent} \\
\text{SetDefaultCurrency} & \\
\text{effect()}
\end{array}
\]
context `SetDefaultCurrency::effect()`
post : Store.allInstances() -> any(true).defaultCurrency = self.currency

CurrencyStatusChange

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{
    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');
    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 attente';
    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.

<table>
<thead>
<tr>
<th>context</th>
<th>Country::nameAndCodesAreUnique: Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Country.allInstances() -&gt; isUnique (name) and</td>
</tr>
<tr>
<td></td>
<td>Country.allInstances() -&gt; isUnique (isoCode2) and</td>
</tr>
<tr>
<td></td>
<td>Country.allInstances() -&gt; isUnique (isoCode3)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>context</th>
<th>Zone::nameAndCountryAndCodeAndCountryAreUnique: Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Zone.allInstances() -&gt; isUnique (Tuple::name, c:country)) and</td>
</tr>
<tr>
<td></td>
<td>Zone.allInstances() -&gt; isUnique (Tuple::code, c:country))</td>
</tr>
</tbody>
</table>

[IC3] A TaxZone is identified by its name.

<table>
<thead>
<tr>
<th>context</th>
<th>TaxZone::nameIsUnique: Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>TaxZone.allInstances() -&gt; isUnique (name)</td>
</tr>
</tbody>
</table>

[IC4] A TaxClass is identified by its name

<table>
<thead>
<tr>
<th>context</th>
<th>TaxClass::nameIsUnique: Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>TaxClass.allInstances() -&gt; isUnique (name)</td>
</tr>
</tbody>
</table>
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:
   \[\Rightarrow \text{NewCountry}\]
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:
   \[\Rightarrow \text{EditCountry}\]
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:

   \[\rightarrow\text{DeleteCountry}\]

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:

   \[\rightarrow\text{NewZone}\]

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:

   \[\rightarrow\text{EditZone}\]

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:
   \[ \rightarrow \text{DeleteZone} \]
3. The system deletes the zone.

Events

NewCountry

```
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

```
context EditCountry::effect()
post:
  c.newName : String
  c.newIsoCode2 : String
  c.newIsoCode3 : String
```

```
\"InitC\"
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

\"InitC\"
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

\"InitC\"
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)
**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**

**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**

**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**

**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** `DeleteZone::effect()`

**Post:**
- `self.zone.name = self.newName` and
- `self.zone.code = self.newCode`
Example test programs

```plaintext
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);
    dos:=new OrderStatus;
    dosl:=new OrderStatusInLanguage(orderStatus:=dos, language:=deutsch);
    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;
  }
}
```
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:
   
   \[
   \text{NewTaxZone}\]

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:
   
   \[ \Rightarrow \text{EditTaxRate} \]

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:
   
   \[ \Rightarrow \text{DeleteTaxRate} \]

3. The system deletes the tax rate.
Events

NewTaxZone

```plaintext
DomainEvent

NewTaxZone

name : String
description : String [0..1]
effect()

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

```plaintext
DomainEvent

EditTaxZone

newName : String
newDescription : String [0..1]
effect()

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

context NewTaxClass::TaxClassDoesNotExist(): Boolean
body :  not TaxClass.allInstances() -> exists (tc | tc.name = self.name)
context NewTaxClass::effect()
post :
  tc.oclsNew() and 
tc.oclsTypeOf(TaxClass) and 
tc.name = self.name and 
tc.description = self.description
**EditTaxClass**

```
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**

```
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**

```
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{
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:='GeneralVAT') 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;
}

/*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

// 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);
  dosInSpanish.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 to 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 selfoclIsTypeOf(Special) then
    if selfoclAsType(Special).specialStatus=Status::enabled and
      selfoclAsType(Special).expiryDate < Now()
      then selfoclAsType(Special).specialPrice
    else set{}
  endif
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

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

context NewProduct::effect()
post:
  p.isNew() and
  p.typeOf(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**

```plaintext
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) 
  self.product.lastModified = Now()
```
DeleteProduct

context DeleteProduct::effect()
post:
  if product@pre.orderLine -> size()=0
  then Product.allInstances->excludes(product@pre)
  else
    psc.oclsNew() and
    psc.oclsTypeOf(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

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

\[
\text{Product} \rightarrow^* \text{Attribute} \rightarrow^* \text{ProductAttribute} \rightarrow \text{Option} \rightarrow^* \text{Value} \rightarrow \text{Downloadable}
\]

- **Downloadable**
  - `filename` : File
  - `expiryDays` : Natural
  - `maximumDownloadCount` : Natural

- **Enumeration**
  - `Sign`: plus, minus

- **Enumeration**
  - `Status`: enabled, disabled

- **Data type**
  - `File`
  - `Money`

\[
\begin{align*}
&\text{context} \quad \text{Language::optionNameIsUnique(): Boolean} \\
&\quad \text{body} : \text{self.hasOptionName} \rightarrow \text{isUnique(optionName)}
\end{align*}
\]

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

**IC2** In each language, each product value has a unique name.
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:  
   \[ \Rightarrow \text{NewProductOption} \]
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:  
   \[ \Rightarrow \text{EditProductOption} \]
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:
   
   \[ \Rightarrow \text{DeleteProductOptionValue} \]

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):
   
   \[ \Rightarrow \text{NewProductAttribute} \]
   
   \[ \Rightarrow \text{NewDownloadableProductAttribute} \]

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

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

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

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

```cpp
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

```cpp
context EditProductOption::effect()
post:
  EditProductOption::effect()
```
```cstl
«InitIC»
context EditProductOption: 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**

```plaintext
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**

```plaintext
context EditProductOptionValue::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
```
CONCEPTUAL MODELING OF INFORMATION SYSTEMS RESEARCH GROUP

```ocelot
«InitC»
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

ExistingValueEvent

DomainEvent

DeleteProductOptionValue

«InitC»
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

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

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

custom
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**

```plaintext
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.expiryDaysnotEmpty() 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

```
<InitC>
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**

- **Downloadable**
  - **ExistingDownloadableEvent**
  - **DomainEvent**
  - **EditDownloadableProductAttribute**
    - **newFilename** : File
    - **newExpiryDays** : Natural
    - **newMaximumDownloadCount** : Natural
    - **effect()**

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

**ProductAttributeStatusChange**

- **ProductAttribute**
  - **ExistingProductAttributeEvent**
  - **DomainEvent**
  - **ProductAttributeStatusChange**
    - **newStatus** : Status
    - **effect()**

**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:=differentName, 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:=differentName, languageOfOption:=english, productOptionNameEvent:=this);
        npo2 may not occur;
        npo3:=new NewProductOption;
        new HasNewOptionName(nameOfOption:=catalanName, languageOfOption:=catalan, productOptionNameEvent:=this);
        new HasNewOptionName(nameOfOption:=catalanName, languageOfOption:=english, productOptionNameEvent:=this);
    }
npo 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: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:english,
        productOptionNameEvent:=this));
    epo may not occur;
    epo2:=new EditProductOption(option:=shirtSize);
    new HasNewOptionName(nameOfOption:=differentName, languageOfOption:english,
        productOptionNameEvent:=this));
    epo2 may not occur;
    epo3:=new EditProductOption(option:=shirtSize);
    new HasNewOptionName(nameOfOption:=catalanName, 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);

  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);

  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:=differentName,
      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);
nov2 may not occur;
npov3:=new NewProductOptionValue(option:=shirtSize);
new HasNewValueName(nameOfValue:=smallInCatalan, languageOfValue:=catalan, productValueNameEvent:=this);
nov3 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);
  specialEditionWithCommentsEdition:=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 DeleteProductAttributeUsedInAnOrder{
  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 Date Time 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:
   
   \[
   \rightarrow \text{NewCategory}
   \]
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:
   
   \[
   \rightarrow \text{EditCategory}
   \]
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:
   \[\rightarrow\text{MoveCategory}\]
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:
   \[\rightarrow\text{DeleteCategory}\]
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:
   \[\rightarrow\text{MoveProduct}\]
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:
   
   \[
   \rightarrow \text{LinkProduct}
   \]
3. The system links the product.

**Events**

**NewCategory**

```
context NewCategory::categoryDoesNotExist(): Boolean
body: Language.allInstances() -> forAll ( 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)
MoveCategory

```plaintext
class MoveCategory
  cyclesDoNotAppear()
    newParent.allParents()->excludes(self.category)
  effect()
    self.category.parent = self.newParent
endclass
```

DeleteCategory

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

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:=trainsInItalian,
            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;
    }
}
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:  
   
   \[ \rightarrow \text{NewSpecial} \]

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:  
   
   \[ \rightarrow \text{EditSpecial} \]

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:

\[ \rightarrow \text{DeleteSpecial} \]

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()
```
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.

Manufacturer

| name : String |
| imagePath : String [0..1] |
| /added : DateTime {<<constant>>} |
| lastModified : DateTime [0..1] |

Language

ManufacturerInLanguage

| url : URL |
| urlClicked : Natural |
| lastClick : DateTime [0..1] |

[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: 
   \[\rightarrow\text{NewManufacturer}\]
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:  
   \[\rightarrow\text{EditManufacturer}\]
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:  
   \[\rightarrow\text{DeleteManufacturer}\]
4. The system deletes the manufacturer and, if requested, changes the status of the products manufactured by it to out of stock.
NewManufacturer

```oclm
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

```
<<interface>>
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
Language.allInstances() ->
forAll(l|
self.hasURL->select(language=l).url=
self.manufacturer.manufacturerInLanguage->
select(language=l).url)

depost:
self.manufacturer.lastModified = Now()
```

### DeleteManufacturer

```
<<interface>>
context DeleteManufacturer::deleteProducts(): Boolean

body:
```
```
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
    englishURL := new URL(url='bookseditorial.com/english');
    bookseditorial := new Manufacturer(name='bookseditorial');
    miEnglish := new ManufacturerInLanguage
        (manufacturer=bookseditorial, language=english);
    miEnglish.url := englishURL;
    miSpanish := new ManufacturerInLanguage
        (manufacturer=bookseditorial, language=spanish);
    miSpanish.url := spanishURL;
    // 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.all Instances->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);

**Structural schema**

`osCommerce` allows administrating banners published in the *online* store.

![Diagram of Banner and BannerGroup entities with associated attributes: Banner: title, url, imagePath, html, expires, added, scheduled, status; BannerGroup: name; BannerHistory: shown, clicked; DataTypes: HtmlText, URL, Date; Status: enabled, disabled.]

**[DR1] Banner::added** is the *DateTime* when the banner was created.

```plaintext
context Banner::added():DateTime
body: Now()
```

**[IC1] A Banner is identified by its title.**

```plaintext
context Banner::titleIsUnique: Boolean
body: Banner.allInstances() -> isUnique(title)
```

**[IC2] A Banner Group is identified by its name.**

```plaintext
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:
   \[\Rightarrow \text{NewBanner} \]
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:
   \[\Rightarrow \text{EditBanner} \]
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:
   \[\rightarrow \text{DeleteBannerGroup}\]
3. The system deletes the banner.

Events

NewBanner

```
DomainEvent

NewBanner

\begin{itemize}
  \item title : String
  \item url : URL [0..1]
  \item imagePath : String
  \item html : HtmlText [0..1]
  \item expires : Date [0..1]
  \item scheduled : Date [0..1]
\end{itemize}
```

context NewBanner::bannerDoesNotExist(): Boolean

\textbf{body}: not Banner.allInstances() | exists (b | b.title = self.title)

context NewBanner::effect()

\textbf{post}:

\begin{itemize}
  \item b.oclIsNew() \textbf{and}
  \item b.oclIsTypeOf(Banner) \textbf{and}
  \item b.title = self.title \textbf{and}
  \item b.url = self.url \textbf{and}
  \item b.imagePath = self.imagePath \textbf{and}
  \item b.html = self.html \textbf{and}
  \item b.expires = self.expires \textbf{and}
  \item b.scheduled = self.scheduled \textbf{and}
  \item b.status = BannerStatus::enabled \textbf{and}
  \item b.bannerGroup = self.bannerGroup
\end{itemize}

NewBannerGroup

```
DomainEvent

NewBannerGroup

\begin{itemize}
  \item name : String
\end{itemize}
```
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

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**

```plaintext
context EditBannerGroup::bannerGroupDoesNotExist():Boolean
   body: (BannerGroup.allInstances - Set{self.bannerGroup}).name->excludes(self.newName)

context EditBannerGroup::effect()
   post : self.bannerGroup.name = self.newName
```

**DeleteBanner**

```plaintext
context DeleteBanner::effect()
   post : not self.banner@pre.oclIsKindOf(OclAny)
```
CONCEPTUAL MODELING OF INFORMATION SYSTEMS RESEARCH GROUP

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 EditBannerGroup(bannerGroup:=bg,newName:='ChristmasSpecials') occurs;
    new EditBannerGroup(bannerGroup:=bg,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.

```text
Newsletter

- title : String
- content : String
- added : DateTime
- status : NewsletterStatus

ProductNotification

- global : Boolean
- explicitRelatedProduct : Product
- explicitNotifications : Newsletter

Product

- notifications : Newsletter

NewsletterStatus

- locked
- unlocked
```

```cstl

```

```cstl

```
**Conceptual Modeling of Information Systems**

**Research Group**

---

**[DR1]** *ProductNotification::notifications* is the set of implied products in the notification.

```plaintext
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.

```plaintext
context Newsletter::added():DateTime
body : Now()
```

**[IC1]** A Newsletter is identified by its title.

```plaintext
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 notification).
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.
   \[\rightarrow UnlockNewsletter\]
2. The system saves the change.

Events

**NewNewsletter**

```
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**

```
context NewProductNotification::ProductNotificationDoesNotExist(): Boolean
body: not Newsletter.allInstances() -> exists (n | n.title = self.title)
context NewProductNotification::effect()
```

```
explicitNotifications + Product
```

```
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

context  EditNewsletter::newsletterIsUnlocked():Boolean
body: self.newsletter.status = Status::unlocked

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 :
  newGlobal : Boolean
  newExplicitNotifications

context  ProductNotification::effect()
context EditProductNotification::effect()
post:
  self.productNotification.global = self.newGlobal and
  self.productNotification.explicitNotifications = self.newExplicitNotifications

DeleteNewsletter

context DeleteNewsletter::newsletterIsUnlocked():Boolean
body: self.newsletter.status = Status::unlocked
context DeleteNewsletter::effect()
post: not self.newsletter@pre.oclIsKindOf(OclAny)

LockNewsletter

context LockNewsletter::newsletterIsNotLocked():Boolean
body: self.newsletter.status <> Status::locked
context LockNewsletter::effect()
post: self.newsletter.status = NewsletterStatus::locked
UnlockNewsletter

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 notification 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 UnlockNewsletter(newsletter:=n1,newTitle:'NewTitle')
            may not occur;
        new UnlockNewsletter(newsletter:=n1)
            may not occur;

        //Valid newsletter editions
        new UnlockNewsletter(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 materialization
    pn1._notifications:=Set{aucaSenyorEsteveBook};
    check consistency;
  }

  test globalNotificationsEnabled{
    pn1.global:=true;
    //We test the derived relationship notifications using materialization
    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)
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.

**Administratate 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:
   \[ \rightarrow \text{NewProductNotificationSubscription} \]
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:
   \[ \rightarrow \text{EditGlobalNotifications} \]
2c. The customer wants to delete a product notification subscription:
   2c1. The customer selects the product:
      \[ \rightarrow \text{DeleteProductNotificationSubscription} \]
   2c2. The use case continues at step 3.

Events

NewCustomer

```
DomainEvent

NewCustomer

eMailAddress : EMail
dateOfBirth : Date [0..1]
phone : String
fax : String [0..1]
primary : Address
newsletter : Boolean
password : String
passwordConfirmation : String
effect()
```
context NewCustomer::customerDoesNotExist(): Boolean
  body : not Customer.allInstances() -> exists (c | c.eMailAddress = self.eMailAddress)

context NewCustomer::passwordCorrect(): Boolean
  body : password = passwordConfirmation

context NewCustomer::firstNameRight(): Boolean
  body : self.primary.firstName.size() >= MinimumValues.firstName

context NewCustomer::lastNameRight(): Boolean
  body : self.primary.lastName.size() >= MinimumValues.lastName

context NewCustomer::dateOfBirthRight(): Boolean
  body : CustomerDetails.dateOfBirth implies
    self.dateOfBirth -> notEmpty() and
    self.dateOfBirth.size() >= MinimumValues.dateOfBirth

context NewCustomer::genderRight(): Boolean
  body : CustomerDetails.gender implies self.gender->notEmpty()

context NewCustomer::suburbRight(): Boolean
  body : CustomerDetails.suburb implies self.suburb->notEmpty()

context NewCustomer::eMailRight(): Boolean
  body : self.eMailAddress.size() >= MinimumValues.eMailAddress

context NewCustomer::streetAddressRight(): Boolean
  body : self.primary.street.size() >= MinimumValues.streetAddress

context NewCustomer::companyRight(): Boolean
  body :
    CustomerDetails.company implies
    self.primary.company -> notEmpty() and
    self.primary.company.size() >= MinimumValues.companyName

context NewCustomer::postCodeRight(): Boolean

context NewCustomer::cityRight(): Boolean
  body : self.primary.city.size() >= MinimumValues.city

context NewCustomer::stateRight(): Boolean
  body :
    CustomerDetails.state implies
    self.primary.state -> notEmpty() and
    self.primary.state.size() >= MinimumValues.state
```plaintext
context NewCustomer::telephoneRight(): Boolean
body : self.telephone.size() >= MinimumValues.telephoneNumber

context NewCustomer::passwordRight(): Boolean
body : self.password.size() >= MinimumValues.password

context NewCustomer::effect()
post :
coclIsNew() and
coclIsTypeOf(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

ExistingCustomerEvent

DomainEvent

Customer

PasswordChange
oldPassword : String
newPassword : String
effect()

context ChangePassword::passwordRight(): Boolean
body : self.password.size() >= MinimumValues.password

context ChangePassword::OldPasswordsCorrect(): Boolean
body : customer.password = self.oldPassword

context ChangePassword::effect()
post : self.customer.password = self.newPassword
```
EditCustomerDetails

Customer

ExistingCustomerEvent  DomainEvent

<table>
<thead>
<tr>
<th>EditCustomerDetails</th>
</tr>
</thead>
<tbody>
<tr>
<td>newGender : Gender [0..1]</td>
</tr>
<tr>
<td>newFirstName : String</td>
</tr>
<tr>
<td>newLastName : String</td>
</tr>
<tr>
<td>newDateOfBirth : Date [0..1]</td>
</tr>
<tr>
<td>newEMailAddress : EMail</td>
</tr>
<tr>
<td>newPhone : String</td>
</tr>
<tr>
<td>newFax : String [0..1]</td>
</tr>
<tr>
<td>newNewsletter : Boolean</td>
</tr>
<tr>
<td>effect()</td>
</tr>
</tbody>
</table>

«InitI»
context EditCustomerDetails::firstNameRight(): Boolean
  body : self.newFirstName.size() >= MinimumValues.firstName

«InitI»
context EditCustomerDetails::lastNameRight(): Boolean
  body : self.newLastName.size() >= MinimumValues.lastName

«InitI»
context EditCustomerDetails::dateOfBirthRight(): Boolean
  body :
    CustomerDetails.dateOfBirth implies
    self.newDateOfBirth->notEmpty()
    self.newDateOfBirth.size() >= MinimumValues.dateOfBirth

«InitI»
context EditCustomerDetails::genderRight(): Boolean
  body : CustomerDetails.gender implies self.newGender->notEmpty()

«InitI»
context EditCustomerDetails::eMailRight(): Boolean
  body : self.newEMailAddress.size() >= MinimumValues.eMailAddress

«InitI»
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

context EditCustomerAddress::AddressOfCustomer(): Boolean
body: self.customer.address -> includes(self.address)

context EditCustomerAddress::firstNameRight(): Boolean
body: self.newAdress.firstName.size() >= MinimumValues.firstName

context EditCustomerAddress::lastNameRight(): Boolean
body: self.newAdress.lastName.size() >= MinimumValues.lastName

context EditCustomerAddress::genderRight(): Boolean
body: CustomerDetails.gender implies self.newAdress.gender->notEmpty()

context EditCustomerAddress::suburbRight(): Boolean
body: CustomerDetails.suburb implies self.newAdress.suburb->notEmpty()

context EditCustomerAddress::streetAddressRight(): Boolean
body: self.newAdress.street.size() >= MinimumValues.streetAddress

context EditCustomerAddress::companyRight(): Boolean
body:
  CustomerDetails.company implies
  self.newAdress.company -> notEmpty() and
  self.newAdress.company.size() >= MinimumValues.companyName

context EditCustomerAddress::postCodeRight(): Boolean

context EditCustomerAddress::cityRight(): Boolean
body: self.newAdress.city.size() >= MinimumValues.city
context EditCustomerAddress::stateRight(): Boolean
    body:
        CustomerDetails.state implies
        self.newAdress.state -> notEmpty() and
        self.newAdress.state.size() >= MinimumValues.state

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

```
| gender : Gender [0..1] |
| firstName : String |
| lastName : String |
| company : String [0..1] |
| street : String |
| suburb : String [0..1] |
| postCode : PostalCode |
| city : String |
| state : String [0..1] |
| effect() |
```
context NewCustomerAddress::suburbRight(): Boolean
body : CustomerDetails.suburb implies self.suburb->notEmpty()

context NewCustomerAddress::streetAddressRight(): Boolean
body : self.primary.street.size() >= MinimumValues.streetAddress

context NewCustomerAddress::companyRight(): Boolean
body :
  CustomerDetails.company implies
  self.primary.company -> notEmpty() and
  self.primary.company.size() >= MinimumValues.companyName

context NewCustomerAddress::postCodeRight(): Boolean

context NewCustomerAddress::cityRight(): Boolean
body : self.primary.city.size() >= MinimumValues.city

context NewCustomerAddress::stateRight(): Boolean
body :
  CustomerDetails.state implies
  self.primary.state -> notEmpty() and
  self.primary.state.size() >= MinimumValues.state

context NewCustomerAddress::addressesHaveZoneIfNeeded(): Boolean
body :
  self.country.zone->size()>0 implies
  (self.state = self.zone.name and
   self.country = self.zone.country)

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

«InitIC»
context DeleteCustomerAddress::AddressOfCustomer(): Boolean
body: self.customer.address -> includes(self.address)

«InitIC»
context DeleteCustomerAddress::AtLeastTwoAddresses(): Boolean
body: self.customer.address.size() >= 2

«InitIC»
context DeleteCustomerAddress::PrimaryAddressCannotBeDeleted(): Boolean
self.address <> self.customer.primary

countext DeleteCustomerAddress::effect()
post: self.customer.address -> excludes(self.address)

PrimaryCustomerAddressChange

«InitIC»
context PrimaryCustomerAddressChange::AddressOfCustomer(): Boolean
body: self.customer.address -> includes(self.address)

countext PrimaryCustomerAddressChange::effect()
post: self.customer.primary = self.address
EditCustomer

Customer
  1
ExistingCustomerEvent  DomainEvent

EditCustomer

newGender : Gender [0..1]
newFirstName : String
newLastName : String
newDateOfBirth : Date [0..1]
newEMailAddress : EMail
newPhone : String
newFax : String [0..1]
newNewsletter : Boolean
newPassword : String
newGlobalNotifications : Boolean

effect()

context EditCustomer::firstNameRight(): Boolean
  body : self.newFirstName.size() >= MinimumValues.firstName

context EditCustomer::lastNameRight(): Boolean
  body : self.newLastName.size() >= MinimumValues.lastName

context EditCustomer::dateOfBirthRight(): Boolean
  body : CustomerDetails.dateOfBirth implies self.newDateOfBirth->notEmpty() and
         self.newDateOfBirth.size() >= MinimumValues.dateOfBirth

context EditCustomer::genderRight(): Boolean
  body : CustomerDetails.gender implies self.newGender->notEmpty()

context EditCustomer::eMailRight(): Boolean
  body : self.newEMailAddress.size() >= MinimumValues.eMailAddress

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
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**

![Diagram of DeleteCustomer](image)

**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**

![Diagram of CustomerStatusChange](image)

**context** CustomerStatusChange::effect()
**post** : self.customer.status = self.newStatus
NewProductNotificationSubscription

$$\text{context NewProductNotificationSubscription::ProductIsUnsubscribed(): Boolean}$$

$$\text{body:}$$


$$\text{context NewProductNotificationSubscription::effect()$$

$$\text{post:}$$

- self.customer.explicitNotifications -> includes(self.newSubscribedProduct)

EditGlobalNotifications

$$\text{context EditGlobalNotifications::effect()$$

$$\text{post:}$$

- 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:=$city, country:=c];
    new NewCustomer(eMailAddress:=e, dateOfBirth:=d, phone:=phone, fax:=e, primary:=a, newsletter:=$newsletter, password:=$password, passwordConfirmation:=$passwordConfirmation) may not occur;
}

//We can easily test the NewCustomer event in different valid or invalid contexts

test validNewCustomer

test validNewCustomer

//Incorrect password confirmation

test invalidNewCustomer

//Incorrect minimumValues

test invalidNewCustomer

test invalidNewCustomer

test invalidNewCustomer

test invalidNewCustomer
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@yyyy.yyy');
d2 := new Date(date:='YY/YY/YYYY');
  new EditCustomerDetails(customer:=john,
    newFirstName:='Johnatan', newLastName:='JR.',
    newEMailAddress:=e2, newDateOfBirth:=d2,
    newPassword:='YYYYYYYYY', newFax:='YYYYYYYYY') occurs;
}
test invalidCustomerDetailsEditions{
  e2 := new EMail(eMail:'');
d2 := new Date;


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:='YYYYYYYY', newFax:='YYYYYYYY') 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:='YYYYYY') may not occur;  
}

}  

testprogram CustomerAddressesManagement{  
//Customer initialization  
catalonia:= new Zone(name:='Catalonia', code:='CAT', country:=spain);  
a:= new Address(country:=spain, zone:=catalonia,  
c := new Customer(address:=a,primary:=a);  
//Other locations to be used  
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',
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
//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',
    city:='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
  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.

<table>
<thead>
<tr>
<th>Review</th>
<th>*</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>review: String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rating: Rating</td>
<td></td>
<td></td>
</tr>
<tr>
<td>added: DateTime</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>lastModified: DateTime</td>
<td>0..1</td>
<td></td>
</tr>
<tr>
<td>timesRead: Natural</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<<enumeration>>

Rating

oneStar
twoStars
threeStars
defStars
fiveStars

[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:
   [\(\text{NewReview}\)]
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:
      \[ \rightarrow \text{LogIn} \]
   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.  
   \[ \rightarrow \text{EditReview} \]
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:  
   \[ \rightarrow \text{DeleteReview} \]
4. The system deletes the review.
Events

NewReview

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. Es un hotel con una distribución 
    extraña 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
end

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)
[DR4] \textbf{Order::phone} of an order is that of its customer.

\begin{verbatim}
context Order::phone():String
body : self.customer.phone
\end{verbatim}

[DR5] \textbf{Order::purchased} is the \textit{DateTime} when the order was created

\begin{verbatim}
context Order::purchased():DateTime
body : Now()
\end{verbatim}

[DR6] \textbf{Order::lastModified} is the last \textit{DateTime} when the status order was modified

\begin{verbatim}
context Order::lastModified():DateTime
body : self.orderStatusChange -> sortedBy(added) -> last().added
\end{verbatim}

[DR7] \textbf{Order::status} is the current status of the order

\begin{verbatim}
context Order::status():OrderStatus
body : self.orderStatusChange -> sortedBy(added) -> last().orderStatus
\end{verbatim}

[DR8] \textbf{Order::total} gives the total amount of an order

\begin{verbatim}
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.percentageIncreaseForLarger Packages/100)
else totalWeight + ShippingAndPackaging.typicalPackageTareWeight
endif
in
totalWithoutShippingCosts +
self.shippingMethod.shippingCosts
(totalWeightIncreased, totalWithoutShippingCosts, quantity) + handlingFee
\end{verbatim}

[DR9] \textbf{OrderStatusChange::added} is the \textit{DateTime} when the change is done.

\begin{verbatim}
context OrderStatusChange::added():DateTime
body : Now()
\end{verbatim}
[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.

   \[ \rightarrow \text{LogIn} \]

**Extensions:**

3a. The customer has a shopping cart from a previous session.

   3a1. The previous shopping cart is restored.

   \[ \rightarrow \text{RestorePreviousShoppingCart} \]

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.

   \[ \rightarrow \text{LogOut} \]

**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.

   \[ \rightarrow \text{SetCurrentLanguage} \]
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.  
   
   \[ \rightarrow \text{SetCurrentCurrency} \]

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:  
   
   \[ \rightarrow \text{LogIn} \]  
   
   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.  
   
   \[ \rightarrow \text{ReadProductInfo} \]  
4. The customer selects a product to buy:  
   
   \[ \rightarrow \text{AddProductToShoppingCart} \]  
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:  
   
   \[ \rightarrow \text{UpdateShoppingCart} \]  
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:
   \[ \text{NewCustomerAddress} \]
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:  
   \[ \text{CancelOrder} \]
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:  
   \[ \text{NewOrderStatus} \]
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:
   \[ \rightarrow EditOrderStatus \]
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:
   \[ \rightarrow DeleteOrderStatus \]
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.
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.

[\(\rightarrow\text{SetCancelledOrderStatus}\)]

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.

[\(\rightarrow\text{SetDefaultOrderStatus}\)]
Events

**NewSession**

- **DomainEvent**
- **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**

- **Session**
- **ExistingSessionEvent**
- **DomainEvent**
- **DeleteSession**
  - context `DeleteSession::effect()`
  - post: `not self.session@pre.oclIsKindOf(OclAny)`
LogIn

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
endif
LogOut

```
context LogOut::customerIsLoggedIn (): Boolean
body : self.session.customer = self.customer

context LogOut::effect()
post : self.session.customer -> isEmpty()
```

SetCurrentLanguage

```
context ChangeCurrentLanguage::effect()
post : self.session.currentLanguage = self.newCurrentLanguage
post : Store.allInstances() -> any(true).switchToDefaultLanguageCurrency and
self.newCurrentCurrentLanguage.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

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

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

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.customerShoppingCart = 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.customerShoppingCart.shoppingCartItem -> includes(sci)
        else
          --The customer does not have a previous shopping cart
          csc.oclIsNew() and
          csc.oclIsTypeOf(CustomerShoppingCart) and
          self.session.customerShoppingCart = csc and
          csc.shoppingCartItem -> includes(sci)
        endif
      endif
    endif
```
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**

```
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**

```
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**

```
context DeleteOrderStatus::IsNotTheCurrentStatusOfAnyOrder(): Boolean
body:
    Order.allInstances() -> forAll (o | o.orderStatusChange -> sortedBy(added)
        -> last().orderStatus <> self.orderStatus)
```
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

OrderConfirmation

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
context OrderConfirmation::ShippingMethodIsEnabled(): Boolean
body : self.shippingMethod.status = Status::enabled

context OrderConfirmation::PaymentMethodIsEnabled(): Boolean
body : self.paymentMethod.status = Status::enabled

context OrderConfirmation::CurrencyIsEnabled(): Boolean
body : self.currency.status = Status::enabled

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()

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.shoppingCart@pre@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))))
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
l1:= 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) may not occur;
}

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;
  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:=l, expectedCurrency:=cu);

test changeCurrentLanguage(switch:=true, newLanguage:=l2, expectedLanguage:=l3, expectedCurrency:=cu2);
testprogram OrderConfirmation{
    // Store initialization
    s:=new Store(name:='FashionTshirts');
    english:=new Language(name:='English', code:='EN');
    s.defaultLanguage:=english;
    s.defaultCurrency:=dollar;
    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,
            languageOfOrderStatus:=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 a store
    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 deleteOrderStatusIfItsTheCurrentStatusOfAnOrder{ // If the order status is the current status of an order, the deletion
        // only changes its status to disabled
Finally, we present a test program that tests a typical scenario of the use case “Place and Order” which is the main functionality of the system from the customer’s point of view.

```plaintext
testprogram PlaceAndOrder{

  // STORE INITIALIZATION
  // Location, currencies and languages
  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
genralRate:= new TaxRate(taxClass:=general, taxZone:=spanishVAT); generalRate.rate:=16;
genralRate.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


APPENDIX A:

osCommerce Conceptual Schema in the USE format

model osCommerce
  -- Enumerations
  enum SortOrder {ascending, descending}
  enum SortField {productName, expectedDate}
  enum Operator {AND, OR}
  enum TransactionMode {test, production}
  enum TransactionMethod {creditCard, eCheck}
  enum PSiGateMode {production, alwaysGood, alwaysDuplicate, alwaysDecline}
  enum PSiGateType {sale, preAuth, postAuth}
  enum PSiGateCollection {local, remote}
  enum SECPayMode {alwaysSuccessful, alwaysFail, production}
  enum Status {enabled, disabled}
  enum USPServer {test, production}
  enum ShippingTableMethod {weight, price}
  enum ProductStatus {inStock, outOfStock}
  enum Sign {plus, minus}
  enum NewsletterStatus {locked, unlocked}
  enum Gender {male, female}
  enum Rating {oneStar, twoStars, threeStars, fourStars, fiveStars}

  -- DataTypes
  class EMail
  attributes
    eMail: String
  end

  class File
  attributes
    fileName: String
  end

  class URL
  attributes
    url: String
  end

  class PostalCode
  attributes
    postalCode: String
  end

  class ShippingTableItem
  attributes
    number: Integer
    cost: Integer
  end

  class DateTime
  attributes
    dateTime: String
  end

  class Date
  attributes
    date: String
  end
--- STRUCTURAL SCHEMA

-- STORE CONFIGURATION

-- Store Data

class Store
attributes
    name: String
    owner: String
    eMailAddress: EMail
    eMailFrom: EMail
    expectedSortOrder: SortOrder
    expectedSortField: SortField
    displayCartAfterAddingProduct: Boolean
    allowGuestToTellAFriend: Boolean
    defaultSearchOperator: Operator
    storeAddressAndPhone: String
    taxDecimalPlaces: Integer
    displayPricesWithTax: Boolean
    switchToDefaultLanguageCurrency: Boolean
end

class NameEMail
end

association store_sendExtraOrderEMail between Store [*] NameEMail[*] role sendExtraOrderEMail
end

association store_defaultLanguage between Store [*] Language[1] role defaultLanguage
end

association store_defaultCurrency between Store [*] Currency[1] role defaultCurrency
end

association store_Country between Store [0..1] Country[1]
end

association store_zone between Store [0..1] Zone[0..1]
end

association store_cancelledStatus between Store [*] role storeOfCancelledStatus
    OrderStatus[1] role cancelledStatus
end

association store_defaultStatus between Store [*] role storeOfDefaultStatus
    OrderStatus[1] role defaultStatus
end

-- Minimum and maximum values

class MinimumValues
attributes
    firstName: Integer
    lastName: Integer
class MaximumValues
  attributes
  addressBookEntries:Integer
end

-- Customer details configuration
class CustomerDetails
  attributes
    gender:Boolean
dateOfBirth:Boolean
company:Boolean
suburb:Boolean
city:Boolean
state:Boolean
end

-- Shipping and Packaging configuration
class ShippingAndPackaging
  attributes
    postCode:PostalCode
  maximumPackageWeight:Integer
typicalPackageTareWeight:Integer
percentageIncreaseForLargerPackages:Integer
end

association shippingAndPackaging_countryOfOrigin between
ShippingAndPackaging [0..1]
Country[1] role countryOfOrigin
end

-- Download configuration
class Download
  attributes
    enableDownload:Boolean
daysExpiryDelay:Integer
maximumNumberOfDownloads:Integer
end

-- Stock configuration
class Stock
  attributes
    checkStockLevel:Boolean
subtractStock:Boolean
allowCheckout:Boolean
stockReOrderLevel:Integer
end

-- Payment methods
abstract class PaymentMethod
attributes
  status:Status
end
association paymentMethod_orderStatus between
  PaymentMethod [*]
  OrderStatus[0..1]
end

association paymentMethod_taxZone between
  PaymentMethod[*]
  TaxZone[0..1]
end

class AuthorizeNet < PaymentMethod
attributes
  username:String
  key:String
  mode:TransactionMode
  method:TransactionMethod
  notification:Boolean
end

class CreditCard < PaymentMethod
attributes
  splitCreditCardToMail:EMail
end

class CashOnDelivery < PaymentMethod
end

class CheckInteger < PaymentMethod
attributes
  makePayableTo:String
end

class Nochex < PaymentMethod
attributes
  eMail:EMail
end

class TwoCheckOut < PaymentMethod
attributes
  login:String
  mode: TransactionMode
  merchantNotification:Boolean
end

abstract class SpecificCurrencyPaymentMethod < PaymentMethod
end

association specificCurrencyPaymentMethod_currency between
  SpecificCurrencyPaymentMethod[*]
  Currency[*]
end

class PSiGate < SpecificCurrencyPaymentMethod
attributes
  merchantID:String
  mode:PSiGateMode
  type:PSiGateType
  creditCardCollection:PSiGateCollection
end

class SECPay < SpecificCurrencyPaymentMethod
attributes
  merchantID:String
  mode:SECPayMode
end

class IPayment < SpecificCurrencyPaymentMethod
attributes
account: Integer
user: String
password: String
end

class PayPal < SpecificCurrencyPaymentMethod
attributes
eMail: EMail
end

class CheckMoney < PaymentMethod
attributes
makePayableTo: String
end

-- Shipping methods
class ShippingMethod
attributes
status: Status
operations
addTaxes(z: Zone, basePrice: Real): Real =
let appliedTaxRates: Set(TaxRate) =
  z.taxZone.taxRate -> select (tr | tr.taxClass = self.taxClass) -> asSet()
in
  let priorities: Set(Integer) =
    if appliedTaxRates -> isEmpty() then oclEmpty(Set(Integer))
    else appliedTaxRates -> sortedBy(priority).priority -> asSet()
  endif
  in
    if priorities -> isEmpty() then basePrice
    else priorities -> iterate (p: Integer; res: Real = 0 |
      res + (((appliedTaxRates -> select (tr | tr.priority = p).rate -> sum())) / 100) + 1) * basePrice)
  endif

shippingCosts(totalWeight: Real, totalPrice: Real, quantity: Integer): Real =
if self.oclIsTypeOf(PerItem) then
  self.oclAsType(PerItem).shippingCosts(totalWeight, totalPrice, quantity)
else
  0.0
endif
end

association shippingMethod_taxClass between
  ShippingMethod[*]
  TaxClass[0..1]
end

class ZoneRates < ShippingMethod
end

association zoneRates_items between
  ZoneRates[*]
  ShippingTableItem[*] role items
end

association zoneRates_country between
  ZoneRates[*]
  Country[*]
end

abstract class SpecificZoneMethod < ShippingMethod
end
association specificZoneMethod_taxZone between
   SpecificZoneMethod[*]
   TaxZone[0..1]
end

abstract class HandlingFeeMethod < ShippingMethod
attributes
   handlingFee:Real
end

class FlatRate < SpecificZoneMethod
attributes
   cost:Real
operations
   shippingCosts3(totalWeight:Real, totalPrice:Real, quantity:Integer): Real =
      self.cost
end

class PerItem < SpecificZoneMethod, HandlingFeeMethod
attributes
   cost:Real
operations
   shippingCosts(totalWeight:Real, totalPrice:Real, quantity:Integer): Real =
      self.cost*quantity
end

class TableRate < SpecificZoneMethod, HandlingFeeMethod
attributes
   method:ShippingTableMethod
operations
   shippingCosts3(totalWeight:Real, totalPrice:Real, quantity:Integer): Real =
      if self.method = #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
end

association tableRate_items between
   TableRate[*]
   ShippingTableItem[*] role items
end

class USPostalService < SpecificZoneMethod, HandlingFeeMethod
attributes
   userID:String
   password:String
   server:USPSServer
operations
   shippingCosts3(totalWeight:Real, totalPrice:Real, quantity:Integer): Real =
      -- we should call USPS service to calculate the shipping costs
end

-- Languages

class Language
attributes
   name: String
   code: String
   directory: String
   sortOrder: Integer
   _prova: Integer
End
association language_defaultCurrency between
  Language[*]
  Currency[0..1] role defaultCurrency
end

-- Currencies
class Currency
attributes
title:String
code:String
symbolLeft:String
symbolRight:String
decimalPlaces:Integer
value:Real
lastUpdate:DateTime
status:Status
end

-- Location & Taxes
class Country
attributes
  name:String
  isoCode2:String
  isoCode3:String
end

class Zone
attributes
  name:String
  code:String
end

association country_zone between
  Country [1]
  Zone[*]
end

class TaxZone
attributes
  name:String
  description:String
end

association zone_taxZone between
  Zone[*]
  TaxZone[*]
end

class TaxClass
attributes
  name:String
  description:String
end

association taxClass_product between
  TaxClass[0..1]
  Product[*]
end

association class TaxRate between
  TaxClass[*]
  TaxZone[*]
attributes
CONCEPTUAL MODELING OF INFORMATION SYSTEMS

-- STORE ADMINISTRATION

class Product
attributes
  status:ProductStatus
  available:Date
  netPrice:Real
  quantityOnHand:Integer
  quantityOrdered:Integer
  modelM:String
  imagePath:String
  added:DateTime
  weight:Real
operations
  specialNetPrice():Real =
    if self.oclIsTypeOf(Special) then
      if self.oclAsType(Special).specialStatus=#enabled
        then self.oclAsType(Special).specialPrice
      else oclEmpty(Set(Real))->any(true)
    endif
    else oclEmpty(Set(Real))->any(true)
  endif
  timesViewed():Integer =
    self.productInLanguage.viewed->sum()
  grossPrice():Real =
    self.addTaxes(Store.allInstances -> any(true).zone, self.netPrice)

  addTaxes(z:Zone,basePrice:Real):Real =
    let appliedTaxRates:Set(TaxRate) =
      TaxRate.allInstances->select(tr | z.taxZone->includes(tr.taxZone)) -> select (tr | tr.taxClass = self.taxClass)
    in
      let priorities:Set(Integer) =
        if appliedTaxRates-> isEmpty() then oclEmpty(Set(Integer))
        else appliedTaxRates -> sortedBy(priority).priority -> asSet()
      endif
      in
        if priorities -> isEmpty() then basePrice
        else priorities -> iterate (p:Integer; res:Real = basePrice | res +
          (((appliedTaxRates -> select (tr | tr.priority = p).rate
            -> sum()) / 100))*res)
        endif
    end

association product_manufacturer between
  Product[*] Manufacturer[0..1]
end

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

associationclass ProductInLanguage between
  Product[*] Language[*]
attributes
name:String
description:String
url:URL
viewed:Integer

-- Product attributes and options
class Option
end
class Value
end

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

associationclass ProductAttribute between
  Product[*]
  Attribute[*]
attributes
  increment:Real
  sign:Sign
  status:Status
end
class Downloadable < ProductAttribute
attributes
  filename:File
  expiryDays:Integer
  maximumDownloadCount:Integer
end
class StringDT
attributes
  string:String
end

associationclass HasOptionName between
  Option[0..1]
  StringDT[1] role optionName
  Language[*] role optionLanguage
end

associationclass HasValueName between
  Value[0..1]
  StringDT[1] role valueName
  Language[*] role valueLanguage
end

-- Product categories
class Category
attributes
  imagePath:String
  sortOrder:Integer
  _subcategories:Integer
  _products:Integer
operations
  subcategories():Integer=self.child->size()
  products():Integer=Category.allInstances
    -> select(c|c.allParents()->includes(self))
    ->union(Set{self}).product->size()
  allParents():Set(Category)=if self.parent.isDefined()
    then self.parent
->union(self.parent.allParents())
else Set{self}
endif-Set{self}
end

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

associationclass HasCategoryName between
    Category[0..1]
    StringDT[*] role categoryName
    Language[*]
end

-- Specials
class Special < Product
attributes
    specialPrice:Real
    expiryDate:Date
    specialLastModified:String
    specialStatus:Status
    dateStatusChanged:DateTime
end

-- Manufacturers
class Manufacturer
attributes
    name:String
    imagePath:String
    lastModified:DateTime
end

associationclass ManufacturerInLanguage between
    Manufacturer[*]
    Language[*]
attributes
    url:URL
    urlClicked:Integer
    lastClick:DateTime
end

-- Banners
class BannerGroup
attributes
    name:String
end
class Banner
attributes
    title:String
    url:URL
    imagePath:String
    html:String
    expires:Date
    scheduled:Date
    statusChanged:DateTime
    status:Status
end

association banner_bannerGroup between
    Banner[*]
    BannerGroup[1]
end

associationclass BannerHistory between
    Banner[*]
Date[*]
attributes
  shown:Integer
  clicked:Integer
end

-- Newsletters

class Newsletter
attributes
  title:String
  content:String
  sent:DateTime
  status:NewsletterStatus
end

class ProductNotification < Newsletter
attributes
  global:Boolean
  _notifications:Set(Product)
operations
  notifications():Set(Product) =
    if self.global then Product.allInstances
    else self.explicitNotifications
    endif
end

association explicitRelatedProduct_explicitNotifications between
  ProductNotification[*] role explicitRelatedProduct
  Product[*] role explicitNotifications
end

-- CUSTOMERS
-- Customers

class Customer
attributes
  gender:Gender
  firstName:String
  lastName:String
  dateOfBirth:Date
  eMailAddress:EMail
  phone:String
  fax:String
  newsletter:Boolean
  password:String
  lastModified:DateTime
  lastLogon:DateTime
  numberOfLogons:Integer
  globalNotifications:Boolean
  status:Status
operations
  notifications():Set(Product) =
    if self.globalNotifications then Product.allInstances
    else self.explicitNotifications
    endif
end

association explicitNotificationSubscriber_explicitNotifications between
  Customer[*] role explicitNotificationSubscriber
  Product[*] role explicitNotifications
end
class Address
attributes
  gender:Gender
  firstName:String
  lastName:String
  company:String
  street:String
  suburb:String
  postCode:PostalCode
  city:String
  state:String
end

association address_zone between
  Address[*]
  Zone[0..1]
end

association address_country between
  Address[*]
  Country[1]
end

association customer_address between
  Customer[*]
  Address[1..*]
end

association primaryAddressCustomer_primary between
  Customer[*] role primaryAddressCustomer
  Address[1] role primary
end

-- ONLINE CATALOG
-- Reviews
class Review
attributes
  review:String
  rating:Rating
  lastModified:DateTime
  timesRead:Integer
end

association review_language between
  Review[*]
  Language[1]
end

association review_product between
  Review[*]
  Product[1]
end

association review_customer between
  Review[*]
  Customer[1]
end

-- Shopping carts
class Session
attributes
  sessionID:Integer
  expiry:DateTime
  ipAddress:String
  timeEntry:DateTime
  timeLastClick:DateTime
  lastPageURL:URL
end
association session_currentLanguage between
  Session[*]
  Language[1] role currentLanguage
end

association session_currentCurrency between
  Session[*]
  Currency[1] role currentCurrency
end

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

class ShoppingCart
end

class AnonymousShoppingCart < ShoppingCart
end

class CustomerShoppingCart < ShoppingCart
end

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

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

class ShoppingCartItem
attributes
  quantity:Integer
operations
  price():Real =
    let netPriceWithSpecial:Real =
      if self.product.specialNetPrice().isUndefined() 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 = #plus
            then increment
          else (-increment)
        endif -> sum() + netPriceWithSpecial
      endif
    end
end

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

association shoppingCartItem_attribute between
  ShoppingCartItem[*]
  Attribute[*]
end
association shoppingCart_shoppingCartItem between
    ShoppingCart[0..1] __
    ShoppingCartItem[1..*] ordered
end

-- Orders
class OrderStatus
    attributes
        status:Status
    end

class Order
    attributes
        delivery:Address
        billing:Address
    operations
        id():Integer=
            if Order.allInstances -> size() = 0 then 0
            else Order.allInstances -> sortedBy(id()) -> last().id() + 1
            endif
        name():String=
            self.customer.firstName
        phone():String=
            self.customer.phone
        eMail():EMail=
            self.customer.eMailAddress
        primary():Address=
            self.customer.primary
        currencyValue():Real=
            self.currency.value
    total():Real=
        let totalWithoutShippingCosts:Real =
            self.orderLine -> collect(finalPrice()*quantity) -> sum()
        in
            let totalWeight:Real =
                self.orderLine -> collect(product.weight*quantity) -> sum()
            in
                let quantity:Integer =
                    self.orderLine.quantity -> sum()
                in
                    let handlingFee:Real =
                        if self.shippingMethod.oclIsKindOf(HandlingFeeMethod)
                            then
                                self.shippingMethod.oclAsType(HandlingFeeMethod).handlingFee
                            else 0.0
                            endif
                        in
                            let totalWeightIncreased:Real =
                                if totalWeight* ((ShippingAndPackaging.allInstances
                                    ->any(true)).percentageIncreaseForLargerPackages/100) >
                                (ShippingAndPackaging.allInstances
                                    ->any(true)).typicalPackageTareWeight
                                then
                                    totalWeight * (1 +totalWeight*
                                        ((ShippingAndPackaging.allInstances
                                            ->any(true)).percentageIncreaseForLargerPackages/100))
                                else totalWeight + (ShippingAndPackaging.allInstances
                                    ->any(true)).typicalPackageTareWeight
                                endif
                            in
                                totalWithoutShippingCosts
                                + self.shippingMethod.shippingCosts(totalWeightIncreased,
                                    totalWithoutShippingCosts, quantity)
                                + handlingFee
                            endif
                    end
                end
            end
        end
    end
association order_customer between
  Order[*]
  Customer[1]
end

association order_shippingMethod between
  Order[*]
  ShippingMethod[1]
end

association order_paymentMethod between
  Order[*]
  PaymentMethod[1]
end

association order_currency between
  Order[*]
  Currency[1]
end

associationclass OrderStatusChange between
  Order[*]
  OrderStatus[1..*] ordered
attributes
  comments:String
end

associationclass OrderStatusInLanguage between
  OrderStatus[*]
  Language[*]
attributes
  name:String
end

class OrderLine
attributes
  quantity:Integer
operations:
  name():String=
    self.product.productInLanguage
    ->select(pil | pil.language = Store.allInstances ->
    any(true).defaultLanguage).name->any(true)
  modelM():String=
    self.product.modelM
  basePrice():Real=
    if self.product.specialNetPrice().isDefined()
    then self.product.specialNetPrice()
    else self.product.netPrice
    endif
  price():Real=
    if self.orderLineAttribute -> isEmpty() then self.basePrice()
    else
      self.orderLineAttribute -> collect
      (if sign() = #plus then increment()
      else (-increment())
      endif) -> sum() + self.basePrice()
    endif
  finalPrice():Real=
    if self.order.billing.zone -> notEmpty() then
self.product.addTaxes(self.order.billing.zone, self.price())
else self.price()
endif

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

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

class OrderLineAttribute
operations
  option():String=
    self.attribute.option.hasOptionName
    -> select (hon | hon.optionLanguage = Store.allInstances
    -> any(true).defaultLanguage).optionName->any(true).string
  value():String=
    self.attribute.value.hasValueName
    -> select (hvn | hvn.valueLanguage = Store.allInstances
    -> any(true).defaultLanguage).valueName->any(true).string
  increment():Real=
    self.attribute.productAttribute
    -> select (pa | pa.product = self.orderLine.product).increment->any(true)
  sign():Sign=
    self.attribute.productAttribute
    -> select (pa | pa.product = self.orderLine.product).sign->any(true)
end

class OrderDownload < OrderLineAttribute
attributes
  downloadCount:Integer
end

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

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

-- BEHAVIOURAL SCHEMA
class Time
end

abstract class Event
attributes
  time:DateTime
operations
  effect()
end

abstract class DomainEvent < Event
end
abstract class ActionRequest < Event end
abstract class Query < ActionRequest end
abstract class SessionEvent end
association sessionEvent_session between
  SessionEvent[*]
  Session[1]
end
class AddProductToShoppingCart < SessionEvent, DomainEvent
  attributes
  quantity:Integer
  operations
effect()
end
association addProductToShoppingCart_attribute between
  AddProductToShoppingCart[*]
  Attribute[*]
end
association addProductToShoppingCart_product between
  AddProductToShoppingCart[*]
  Product[1]
end
class AddressBookEntriesMaximumChange < DomainEvent
  attributes
  newMaximum:Integer
  operations
effect()
end
class AllowCheckoutStockConfigurationChange < DomainEvent
  attributes
  newValue:Boolean
  operations
effect()
end
abstract class StoreEvent
  operations
    myStore():Store=Store.allInstances->any(true)
end
class AllowGuestToTellAFriendChange < DomainEvent, StoreEvent
  attributes
  newAllowGuestToTellAFriend:Boolean
  operations
effect()
end
abstract class ExistingProductAttributeEvent end
association existingProductAttributeEvent_productAttribute between
  ExistingProductAttributeEvent[*]
  ProductAttribute[0..1]
end
class AttributeChange < DomainEvent, ExistingProductAttributeEvent
  operations
    effect()
  end

association attributeChange_Value between
  AttributeChange[*]
Value[1] role newValue
end

association attributeChange_Option between
  AttributeChange[*]
Option[1] role newOption
end

abstract class ExistingOrderEvent
end

association existingOrderEvent_Order between
  ExistingOrderEvent[*]
Order[1]
end

class CancelOrder < DomainEvent, ExistingOrderEvent
  operations
    effect()
  end

class CheckLevelStockConfigurationChange < DomainEvent
  attributes
    newValue:Boolean
  operations
    effect()
  end

class CityMinimumChange < DomainEvent
  attributes
    newMinimum:Integer
  operations
    effect()
  end

abstract class ExistingBannerEvent
end

association existingBannerEvent_banner between
  ExistingBannerEvent[*]
Banner[0..1]
end

class ClickBanner < DomainEvent, ExistingBannerEvent
  operations
    effect()
  end

abstract class ExistingManufacturerEvent
end

association existingManufacturerEvent_banner between
  ExistingManufacturerEvent[*]
Manufacturer[0..1]
end

class ClickManufacturer < DomainEvent, ExistingManufacturerEvent
  operations
    effect()
association clickManufacturer_language between
    ClickManufacturer[*]
    Language[1]
end

class CompanyCustomerDetailChange < DomainEvent
attributes
    newValue:Boolean
operations
    effect()
end

class CompanyNameMinimumChange < DomainEvent
attributes
    newMinimum:Integer
operations
    effect()
end

class CountryChange < DomainEvent,StoreEvent
operations
    effect()
end
association countryChange_country between
    CountryChange[*]
end

class CountryShippingConfigurationChange < DomainEvent
operations
    effect()
end
association countryShippingConfigurationChange_country between
    CountryShippingConfigurationChange[*]
end

class CreditCardNumberMinimumChange < DomainEvent
attributes
    newMinimum:Integer
operations
    effect()
end

class CreditCardOwnerNameMinimumChange < DomainEvent
attributes
    newMinimum:Integer
operations
    effect()
end

abstract class ExistingCurrencyEvent
end
association existingCurrencyEvent_currency between
    ExistingCurrencyEvent[*]
    Currency[0..1]
end
class CurrencyStatusChange < DomainEvent, ExistingCurrencyEvent
attributes
  newStatus:Status
operations
  effect()
end

abstract class ExistingCustomerEvent
end

association existingCustomerEvent_customer between
  ExistingCustomerEvent[*]
  Customer[0..1]
end

class CustomerStatusChange < DomainEvent, ExistingCustomerEvent
attributes
  newStatus:Status
operations
  effect()
end

class DateOfBirthCustomerDetailChange < DomainEvent
attributes
  newValue:Boolean
operations
  effect()
end

class DateOfBirthMinimumChange < DomainEvent
attributes
  newMinimum:Integer
operations
  effect()
end

class DaysExpiryDelayDownloadConfigurationChange < DomainEvent
attributes
  newValue:Integer
operations
  effect()
end

class DefaultSearchOperatorChange < DomainEvent, StoreEvent
attributes
  newDefaultSearchOperator:Operator
operations
  effect()
end

class DeleteBanner < DomainEvent, ExistingBannerEvent
operations
  effect()
end

abstract class ExistingBannerGroupEvent
end

association existingBannerGroupEvent_bannerGroup between
  ExistingBannerGroupEvent[*]
  BannerGroup[0..1]
end

class DeleteBannerGroup < DomainEvent, ExistingBannerGroupEvent
operations
  effect()
end
abstract class ExistingCategoryEvent
end

association existingCategoryEvent_category between
   ExistingCategoryEvent[*]
   Category[0..1]
end

class DeleteCategory < DomainEvent, ExistingCategoryEvent
operations
   effect()
   allChilds(cat:Category):Set(Category) = if cat.child->isEmpty()
      then oclEmpty(Set(Category))
      else cat.child->iterate(c;
         a:Set(Category) = cat.child | a->union(self.allChilds(c))
      )
end

abstract class ExistingCountryEvent
end

association existingCountryEvent_country between
   ExistingCountryEvent[*]
   Country[0..1]
end

class DeleteCountry < DomainEvent, ExistingCountryEvent
operations
   effect()
end

class DeleteCurrency < DomainEvent, ExistingCurrencyEvent
operations
   effect()
end

class DeleteCustomer < DomainEvent, ExistingCustomerEvent
operations
   effect()
end

abstract class ExistingAddressEvent
end

association existingAddressEvent_address between
   ExistingAddressEvent[*]
   Address[1]
end

class DeleteCustomerAddress < DomainEvent, ExistingCustomerEvent, ExistingAddressEvent
operations
   effect()
end

abstract class ExistingLanguageEvent
end

association existingLanguageEvent_language between
   ExistingLanguageEvent[*]
   Language[0..1]
end
class DeleteLanguage < DomainEvent, ExistingLanguageEvent
  operations
    effect()
  end

class DeleteManufacturer < DomainEvent, ExistingManufacturerEvent
  attributes
deleteProds:Boolean
  operations
    effect()
  end

abstract class ExistingNewsletterEvent
end

association existingNewsletterEvent_newsletter between
  ExistingNewsletterEvent[*]
  Newsletter[0..1]
end

class DeleteNewsletter < DomainEvent, ExistingNewsletterEvent
  operations
    effect()
  end

abstract class ExistingOrderStatusEvent
end

association existingOrderStatusEvent_orderStatus between
  ExistingOrderStatusEvent[*]
  OrderStatus[0..1]
end

class DeleteOrderStatus < DomainEvent, ExistingOrderStatusEvent
  operations
    effect()
  end

abstract class ExistingProductEvent
end

association existingProductEvent_product between
  ExistingProductEvent[*]
  Product[0..1]
end

class DeleteProduct < DomainEvent, ExistingProductEvent
  operations
    effect()
  end

class DeleteProductAttribute < DomainEvent, ExistingProductAttributeEvent
  operations
    effect()
  end

class DeleteProductNotificationSubscription < DomainEvent, ExistingCustomerEvent
  operations
    effect()
  end

association deleteProductNotificationSubscription_product between
  DeleteProductNotificationSubscription[*]
Product[1] role deletedSubscribedProduct
end

abstract class ExistingOptionEvent
end

association existingOptionEvent_option between
   ExistingOptionEvent[*]
   Option[0..1]
end

class DeleteProductOption < DomainEvent, ExistingOptionEvent
operations
   effect()
end

abstract class ExistingValueEvent
end

association existingValueEvent_option between
   ExistingValueEvent[*]
   Value[0..1]
end

class DeleteProductOptionValue < DomainEvent, ExistingValueEvent
operations
   effect()
end

abstract class ExistingReviewEvent
end

association existingReviewEvent_review between
   ExistingReviewEvent[*]
   Review[0..1]
end

class DeleteReview < DomainEvent, ExistingReviewEvent
operations
   effect()
end

abstract class ExistingSessionEvent
end

association existingSessionEvent_Session between
   ExistingSessionEvent[*]
   Session[0..1]
end

class DeleteSession < DomainEvent, ExistingSessionEvent
operations
   effect()
end

abstract class ExistingSpecialEvent
end

association existingSpecialEvent_special between
   ExistingSpecialEvent[*]
   Special[0..1]
end
class DeleteSpecial < DomainEvent, ExistingSpecialEvent
operations
effect()
end

abstract class ExistingTaxClassEvent
end

association existingTaxClassEvent_taxClass between
   ExistingTaxClassEvent[*]
   TaxClass[0..1]
end

class DeleteTaxClass < DomainEvent, ExistingTaxClassEvent
operations
effect()
end

abstract class ExistingTaxRateEvent
end

association existingTaxRateEvent_taxRate between
   ExistingTaxRateEvent[*]
   TaxRate[0..1]
end

class DeleteTaxRate < DomainEvent, ExistingTaxRateEvent
operations
effect()
end

abstract class ExistingTaxZoneEvent
end

association existingTaxZoneEvent_taxZone between
   ExistingTaxZoneEvent[*]
   TaxZone[0..1]
end

class DeleteTaxZone < DomainEvent, ExistingTaxZoneEvent
operations
effect()
end

abstract class ExistingZoneEvent
end

association existingZoneEvent_zone between
   ExistingZoneEvent[*]
   Zone[0..1]
end

class DeleteZone < DomainEvent, ExistingZoneEvent
operations
effect()
end

class DisplayCartAfterAddingProductChange < DomainEvent, StoreEvent
attributes
   newDisplayCartAfterAddingProduct:Boolean
operations
effect()
end

class DisplayPricesWithTaxChange < DomainEvent, StoreEvent
attributes
   newDisplayPricesWithTax:Boolean
operations
abstract class EditPaymentMethodEvent
attributes
  status: Status
end

association editPaymentMethodEvent_taxZone between EditPaymentMethodEvent[*]
  TaxZone[0..1]
end

association editPaymentMethodEvent_orderStatus between EditPaymentMethodEvent[*]
  OrderStatus[0..1]
end

class EditAuthorizeNetPaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
  newUsername: String
  newKey: String
  newMethod: TransactionMethod
  newNotification: Boolean
operations
  effect()
end

class EditBanner < DomainEvent, ExistingBannerEvent
attributes
  newTitle: String
  newUrl: URL
  newPath: String
  newHtml: String
  newExpires: Date
  newScheduled: Date
  newStatus: Status
operations
  effect()
end

association editBanner_bannerGroup between EditBanner[*]
  BannerGroup[1] role newBannerGroup
end

class EditBannerGroup < DomainEvent, ExistingBannerGroupEvent
attributes
  newName: String
operations
  effect()
end

class EditCashOnDeliveryPaymentMethod < DomainEvent, EditPaymentMethodEvent
operations
  effect()
end

abstract class CategoryNameEvent
end

association class HasNewName between CategoryNameEvent[*]
  Language[*] role languageOfCategory
  StringDT[1] role name
class EditCategory < DomainEvent, ExistingCategoryEvent, CategoryNameEvent
attributes
  imagePath: String
  sortOrder: Integer
operations
effect()
end

association editCategory_category between
  EditCategory[*] role newParent
end

class EditCheckMoneyPaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
  newMakePayableTo: String
operations
effect()
end

class EditCountry < DomainEvent, ExistingCountryEvent
attributes
  newName: String
  newIsoCode2: String
  newIsoCode3: String
operations
effect()
end

class EditCreditCardPaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
  newSplitCreditCardToMail: EMail
operations
effect()
end

class EditCurrency < DomainEvent, ExistingCurrencyEvent
attributes
  newTitle: String
  newCode: String
  newSymbolLeft: String
  newSymbolRight: String
  newDecimalPlaces: Integer
  newValue: Real
operations
effect()
end

class EditCustomer < DomainEvent, ExistingCustomerEvent
attributes
  newGender: Gender
  newFirstName: String
  newLastName: String
  newDateOfBirth: Date
  newEMailAddress: EMail
  newPhone: String
  newFax: String
  newNewsletter: Boolean
  newPassword: String
  newGlobalNotifications: Boolean
operations
effect()
end

class EditCustomerAddress < DomainEvent, ExistingCustomerEvent, ExistingAddressEvent
attributes

newAddress:Address
operations
  effect()
end

class EditCustomerDetails < DomainEvent, ExistingCustomerEvent
attributes
  newGender:Gender
  newFirstName:String
  newLastName:String
  newDateOfBirth:Date
  newEmailAddress:EMail
  newPhoneNumber:String
  newFax:String
  newNewsletter:Boolean
operations
  effect()
end

abstract class ExistingDownloadableEvent
end

association existingDownloadableEvent_Downloadable between
  ExistingDownloadableEvent[*]
  Downloadable[1]
end

class EditDownloadableAttribute < DomainEvent, ExistingDownloadableEvent
attributes
  newFilename:File
  newExpiryDays:Integer
  newMaximumDownloadCount:Integer
operations
  effect()
end

abstract class ShippingMethodEvent
attributes
  status:Status
end

association ShippingMethodEvent_taxClass between
  ShippingMethodEvent[*]
  TaxClass[0..1]
end

abstract class SpecificZoneShippingMethodEvent < ShippingMethodEvent
end

association SpecificZoneShippingMethodEvent_taxZone between
  SpecificZoneShippingMethodEvent[*]
  TaxZone[0..1]
end

class EditFlatRateShippingMethod < DomainEvent, SpecificZoneShippingMethodEvent
attributes
  newCost:Real
operations
  effect()
end

class EditGlobalNotifications < DomainEvent, ExistingCustomerEvent
attributes
  newGlobalNotifications:Boolean
operations
class EditIPaymentPaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
  newAccount:Integer
  newUser:String
  newPassword:String
operations
  effect()
end

class EditLanguage < DomainEvent, ExistingLanguageEvent
attributes
  newName:String
  newCode:String
operations
  effect()
end

association editLanguage_currency between
  EditLanguage[*]
  Currency[0..1] role newDefaultCurrency
end

abstract class ManufacturerURLEvent
end

associationclass HasURL between
  ManufacturerURLEvent[*]
  Language[*] role languageOfURL
  URL[1] role url
end

class EditManufacturer < DomainEvent, ExistingManufacturerEvent, ManufacturerURLEvent
attributes
  imagePath:String
  name:String
operations
  effect()
end

class EditNewsletter < DomainEvent, ExistingNewsletterEvent
attributes
  newTitle:String
  newContent:String
operations
  effect()
end

class EditNochexPaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
  newEMail:EMail
operations
  effect()
end

abstract class OrderStatusNameEvent
end

associationclass HasOrderStatusName between
  OrderStatusNameEvent[*]
  Language[*] role languageOfOrderStatus
  StringDT[1] role orderStatusName
end

class EditOrderStatus < DomainEvent, ExistingOrderStatusEvent, OrderStatusNameEvent
class EditPayPalPaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
  newEMail:EMail
operations
  effect()
end

association editPayPalMethod_currency between
  EditPayPalPaymentMethod[*]
  Currency[0..1]
end

abstract class HandlingFeeMethodEvent
attributes
  handlingFee:Real
operations
  effect()
end

class EditPerItemShippingMethod < DomainEvent, SpecificZoneShippingMethodEvent,
  HandlingFeeMethodEvent
attributes
  newCost:Real
operations
  effect()
end

abstract class ProductNameEvent
end

association class HasNewProductName between
  ProductNameEvent[*]
  Language[*] role languageOfProduct
  StringDT[1] role nameOfProduct
end

class EditProduct < DomainEvent, ExistingProductEvent, ProductNameEvent
attributes
  status:ProductStatus
  available:Date
  netPrice:Real
  quantityOnHand:Integer
  modelM:String
  imagePath:String
  weight:Real
operations
  effect()
end

association editProduct_manufacturer between
  EditProduct[*]
  Manufacturer[0..1]
end

association editProduct_category between
  EditProduct[*]
  Category[*]
end

association editProduct_taxClass between
  EditProduct[*]
  TaxClass[0..1]
end
class EditProductNotification < DomainEvent
attributes
    newGlobal:Boolean
operations
    effect()
end

association editProductNotification_product between EditProductNotification[*]
    Product[*] role newExplicitNotifications
end

association editProductNotification_productNotification between EditProductNotification[*]
    ProductNotification[1]
end

abstract class ProductOptionNameEvent
end

association class HasNewOptionName between ProductOptionNameEvent[*]
    Language[*] role languageOfOption
    StringDT[1] role nameOfOption
end

class EditProductOption < DomainEvent, ExistingOptionEvent, ProductOptionNameEvent
operations
    effect()
end

abstract class ProductValueNameEvent
end

association class HasNewValueName between ProductValueNameEvent[*]
    Language[*] role languageOfValue
    StringDT[1] role nameOfValue
end

class EditProductOptionValue < DomainEvent, ExistingValueEvent, ProductValueNameEvent
operations
    effect()
end

association editProductOptionValue_Option between EditProductOptionValue[*]
    Option[1..*]
end

class EditPSiGatePaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
    newMerchantID:String
    newMode:PSiGateMode
    newType:PSiGateType
    newCreditCardCollection:PSiGateCollection
operations
    effect()
end

association editPSiGatePaymentMethod_currency between EditPSiGatePaymentMethod[*]
    Currency[0..1]
end

class EditReview < DomainEvent, ExistingReviewEvent
attributes
newReview:String
newRating:Rating
operations
effect()
end

association editReview_Language between
   EditReview[*]
   Language[1] role newLanguage
end

association editReview_Product between
   EditReview[*]
   Product[1] role newProduct
end

association editReview_Customer between
   EditReview[*]
   Customer[1] role newCustomer
end

class EditSECPaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
   newMerchantID:String
   newMode:SECPayMode
operations
effect()
end

association editSECPaymentMethod_currency between
   EditSECPaymentMethod[*]
   Currency[0..1]
end

class EditSpecial < DomainEvent, ExistingSpecialEvent
attributes
   newSpecialPrice:Real
   newExpiryDate:Date
   newStatus:Status
operations
effect()
end

class EditTableRateShippingMethod < DomainEvent, SpecificZoneShippingMethodEvent,
HandlingFeeMethodEvent
attributes
   newMethod:ShippingTableMethod
operations
effect()
end

association editTableRateShippingMethod_newItems between
   EditTableRateShippingMethod[*]
   ShippingTableItem[*] role newItems
end

class EditTaxClass < DomainEvent, ExistingTaxClassEvent
attributes
   newName:String
   newDescription:String
operations
effect()
end

class EditTaxRate < DomainEvent, ExistingTaxRateEvent
attributes
newRate:Integer
newPriority:Integer
newDescription:String
operations
effect()
end

association editTaxRate_taxZone between
EditTaxRate[*] TaxZone[1] role newTaxZone
end

association editTaxRate_taxClass between
EditTaxRate[*] TaxClass[1] role newTaxClass
end

class EditTaxZone < DomainEvent, ExistingTaxZoneEvent
attributes
   newName:String
   newDescription:String
operations
effect()
end

association editTaxZone_newZones between
EditTaxZone[*] Zone[*] role newZones
end

class EditTwoCheckOutPaymentMethod < DomainEvent, EditPaymentMethodEvent
attributes
   newLogin:String
   newMode:TransactionMode
   newMerchantNotification:Boolean
operations
effect()
end

class EditUSPostalServiceShippingMethod < DomainEvent, SpecificZoneShippingMethodEvent, HandlingFeeMethodEvent
attributes
   newUserID:String
   newPassword:String
   newServer:USPSServer
operations
effect()
end

class EditZone < DomainEvent, ExistingZoneEvent
attributes
   newName:String
   newCode:String
operations
effect()
end

class EditZoneRatesShippingMethod < DomainEvent, ShippingMethodEvent
operations
effect()
end

association editZoneRatesShippingMethod_country between
EditZoneRatesShippingMethod[*] Country[*]
end
association editZoneRatesShippingMethod_mewItems between
 EditZoneRatesShippingMethod[*]
   ShippingTableItem[*] role newItems
end

class EMailAddressChange < DomainEvent, StoreEvent
attributes
  newEmailAddress:EMail
operations
  effect()
end

class EMailAddressMinimumChange < DomainEvent
attributes
  newMinimum:Integer
operations
  effect()
end

class EMailFromChange < DomainEvent, StoreEvent
attributes
  newEmailFrom:EMail
operations
  effect()
end

class EnableDownloadConfigurationChange < DomainEvent
attributes
  newValue:Boolean
operations
  effect()
end

class ExpectedSortFieldChange < DomainEvent, StoreEvent
attributes
  newExpectedSortField:SortField
operations
  effect()
end

class ExpectedSortOrderChange < DomainEvent, StoreEvent
attributes
  newExpectedSortOrder:SortOrder
operations
  effect()
end

class FirstNameMinimumChange < DomainEvent
attributes
  newMinimum:Integer
operations
  effect()
end

class GenderCustomerDetailChange < DomainEvent
attributes
  newValue:Boolean
operations
  effect()
end

class IncrementAndSignAttributeChange < DomainEvent, ExistingProductAttributeEvent
attributes
  newIncrement:Real
newSign:Sign
operations
effect()
end

class InstallAuthorizeNetPaymentMethod < DomainEvent
operations
effect()
end

class InstallCashOnDeliveryPaymentMethod < DomainEvent
operations
effect()
end

class InstallCheckMoneyPaymentMethod < DomainEvent
operations
effect()
end

class InstallCreditCardPaymentMethod < DomainEvent
operations
effect()
end

class InstallFlatRateShippingMethod < DomainEvent
operations
effect()
end

class InstallIPaymentPaymentMethod < DomainEvent
operations
effect()
end

class InstallNochexPaymentMethod < DomainEvent
operations
effect()
end

class InstallPayPalPaymentMethod < DomainEvent
operations
effect()
end

class InstallPerItemShippingMethod < DomainEvent
operations
effect()
end

class InstallPSiGatePaymentMethod < DomainEvent
operations
effect()
end

class InstallSECPaymentMethod < DomainEvent
operations
effect()
end

class InstallTableRateShippingMethod < DomainEvent
operations
effect()
end

class InstallTwoCheckOutPaymentMethod < DomainEvent
operations
class InstallUSPostalServiceShippingMethod < DomainEvent
  operations
    effect()
  end

class InstallZoneRatesShippingMethod < DomainEvent
  operations
    effect()
  end

class LastNameMinimumChange < DomainEvent
  attributes
    newMinimum: Integer
  operations
    effect()
  end

class LinkProduct < DomainEvent, ExistingProductEvent
  operations
    effect()
  end

  association linkProduct_category between
    LinkProduct[*]
    Category[1] role newCategory
  end

class LockNewsletter < DomainEvent, ExistingNewsletterEvent
  operations
    effect()
  end

class LogIn < DomainEvent, ExistingCustomerEvent
  operations
    effect()
  end

  association logIn_session between
    LogIn[*]
    Session[0..1]
  end

class LogOut < DomainEvent, ExistingCustomerEvent, ExistingSessionEvent
  operations
    effect()
  end

class NameChange < DomainEvent, StoreEvent
  attributes
    newName: String
  operations
    effect()
  end

class MaximumNumberDownloadConfigurationChange < DomainEvent
  attributes
    newMaximum: Integer
  operations
    effect()
  end

class MaximumPackageWeightShippingConfigurationChange < DomainEvent
  attributes
class MoveCategory < DomainEvent, ExistingCategoryEvent
operations
effect()
end

association moveCategory_newParent between
  MoveCategory[*]
  Category[0..1] role newParent
end

class MoveProduct < DomainEvent, ExistingProductEvent
operations
effect()
end

association moveProduct_oldCategory between
  MoveProduct[*]
  role moveProductOfOldCategory
  Category[1] role oldCategory
end

association moveProduct_newCategory between
  MoveProduct[*]
  role moveProductOfNewCategory
  Category[1] role newCategory
end

class NewBanner < DomainEvent
attributes
title:String
url:URL
imagePath:String
html:String
expires:Date
scheduled:Date
operations
effect()
end

class NewBannerGroup < DomainEvent
attributes
  name:String
operations
effect()
end

association newBanner_bannerGroup between
  NewBanner[*]
  BannerGroup[1];
end

class NewCategory < DomainEvent, CategoryNameEvent
attributes
  imagePath:String
  sortOrder:Integer
operations
effect()
end

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

class NewCountry < DomainEvent
attributes
  name: String
  isoCode2: String
  isoCode3: String
operations
  effect()
end

class NewCurrency < DomainEvent
attributes
  title: String
  code: String
  symbolLeft: String
  symbolRight: String
  decimalPlaces: Integer
  value: Real
operations
  effect()
end

class NewCustomer < DomainEvent
attributes
  dateOfBirth: Date
  eMailAddress: EMail
  phone: String
  fax: String
  newsletter: Boolean
  password: String
  passwordConfirmation: String
  primary: Address
  customerCreated: Customer
operations
  effect()
end

class NewCustomerAddress < DomainEvent, ExistingCustomerEvent
attributes
  gender: Gender
  firstName: String
  lastName: String
  company: String
  street: String
  suburb: String
  postCode: PostalCode
  city: String
  state: String
  answer: Address
operations
  effect()
end

association newCustomerAddress_zone between
NewCustomerAddress[*]
  Zone[0..1]
end

association newCustomerAddress_country between
NewCustomerAddress[*]
  Country[1]
end

class NewDownloadableProductAttribute < DomainEvent, ExistingProductEvent
attributes
  increment: Real
  sign: Sign
  filename: File
expiryDays:Integer
maximumDownloadCount:Integer
operations
effect()
end

association newDownloadableProductAttribute_option between
   NewDownloadableProductAttribute[*]
   Option[1]
end

association newDownloadableProductAttribute_value between
   NewDownloadableProductAttribute[*]
   Value[1]
end

class NewLanguage < DomainEvent
attributes
   newName:String
   newCode:String
operations
effect()
end

association NewLanguage_currency between
   NewLanguage[*]
   Currency[0..1] role defaultCurrency
end

class NewManufacturer < DomainEvent, ManufacturerURLEvent
attributes
   imagePath:String
   name:String
operations
effect()
end

class NewNewsletter < DomainEvent
attributes
   title:String
   content:String
operations
effect()
end

class NewOrderStatus < DomainEvent, OrderStatusNameEvent
attributes
   name:String
   createdOrderStatus:OrderStatus;
operations
effect()
end

class NewProduct < DomainEvent, ProductNameEvent
attributes
   status:ProductStatus
   available:Date
   netPrice:Real
   quantityOnHand:Integer
   modelM:String
   imagePath:String
   weight:Real
operations
effect()
end
association newProduct_manufacturer between
    NewProduct[*]
    Manufacturer[0..1]
end

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

association newProduct_taxClass between
    NewProduct[*]
    TaxClass[0..1]
end

class NewProductAttribute < DomainEvent, ExistingProductEvent
attributes
    increment:Real
    sign:Sign
operations
    effect()
end

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

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

class NewProductNotification < DomainEvent
attributes
    title:String
    content:String
    global:Boolean
operations
    effect()
end

association newProductNotification_product between
    NewProductNotification[*]
    Product[*] role explicitNotifications
end

class NewProductNotificationSubscription < DomainEvent, ExistingCustomerEvent
operations
    effect()
end

association newProductNotificationSubscription_product between
    NewProductNotificationSubscription[*]
    Product[1] role newSubscribedProduct
end

class NewProductOption < DomainEvent, ProductOptionNameEvent
operations
    effect()
end

class NewProductOptionValue < DomainEvent, ProductValueNameEvent
operations
    effect()
end
association newProductOptionValue_option between NewProductOptionValue[*] Option[1..*] role option
end

class NewReview < DomainEvent attributes
  review:String
  rating:Rating
  createdReview:Review
operations
effect()
end

end

end

end

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

association newSession_currentCurrency between NewSession[*] Currency[1] role currentCurrency
end

association newSession_currentLanguage between NewSession[*] Language[1] role currentLanguage
end

class NewSpecial < DomainEvent attributes
  specialPrice:Real
  expiryDate:Date
  status:Status
operations
effect()
end

association newSpecial_product between NewSpecial[*] Product[0..1]
end

class NewTaxClass < DomainEvent attributes
  name:String
  description:String
operations
effect()
end
class NewTaxRate < DomainEvent
  attributes
    rate:Integer
    priority:Integer
    description:String
  operations
    effect()
  end

association newTaxRate_taxZone between
  NewTaxRate[*]
  TaxZone[1]
end

association newTaxRate_taxClass between
  NewTaxRate[*]
  TaxClass[1]
end

class NewTaxZone < DomainEvent
  attributes
    name:String
    description:String
  operations
    effect()
  end

association newTaxZone_mewZones between
  NewTaxZone[*]
  Zone[*]
end

class NewZone < DomainEvent
  attributes
    name:String
    code:String
  operations
    effect()
  end

association newZone_country between
  NewZone[*]
  Country[0..1]
end

class OrderConfirmation < DomainEvent
  attributes
    delivery:Address
    billing:Address
    creditCardType:String
    creditCardOwner:String
    creditCardNumber:String
    creditCardExpires:Date
    comments:String
    orderCreated:Order
  operations
    effect()
  end

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

association orderConfirmation_paymentMethod between
   OrderConfirmation[*]
   PaymentMethod[*]
end

association orderConfirmation_currency between
   OrderConfirmation[*]
   Currency[*]
end

class OwnerChange < DomainEvent, StoreEvent
attributes
   newOwner:String
operations
   effect()
end

class PasswordChange < DomainEvent, ExistingCustomerEvent
attributes
   oldPassword:String
   newPassword:String
operations
   effect()
end

class PasswordMinimumChange < DomainEvent
attributes
   newMinimum:Integer
operations
   effect()
end

class PercentageIncreaseForLargerPackagesShippingConfigurationChange < DomainEvent
attributes
   newPercentage:Real
operations
   effect()
end

class PostCodeMinimumChange < DomainEvent
attributes
   newMinimum:Integer
operations
   effect()
end

class PostCodeShippingConfigurationChange < DomainEvent
attributes
   newPostCode:PostalCode
operations
   effect()
end

class PrimaryCustomerAddressChange < DomainEvent, ExistingAddressEvent, ExistingCustomerEvent
operations
   effect()
end

class ProductAttributeStatusChange < DomainEvent, ExistingProductAttributeEvent
attributes
   newStatus:Status
operations
class ProductDownload < DomainEvent, ExistingCustomerEvent, ExistingProductEvent
  operations
  effect()
end

association productDownload_downloadable between
  ProductDownload[*]
  Downloadable[1]
end

class ProductOptionAttributeChange < DomainEvent, ExistingProductAttributeEvent
  operations
  effect()
end

association productOptionAttributeChange_option between
  ProductOptionAttributeChange[*]
  Option[1]
end

class ProductValueAttributeChange < DomainEvent, ExistingProductAttributeEvent
  operations
  effect()
end

association productValueAttributeChange_value between
  ProductValueAttributeChange[*]
  Value[1]
end

class ProductStatusChange < DomainEvent, ExistingProductEvent
  attributes
    newStatus:ProductStatus
  operations
    effect()
end

class ReadProductInfo < DomainEvent, ExistingProductEvent
  operations
  effect()
end

association readProductInfo_language between
  ReadProductInfo[*]
  Language[1]
end

class ReadReview < DomainEvent, ExistingReviewEvent
  operations
  effect()
end

class ReorderLevelStockConfigurationChange < DomainEvent
  attributes
    newValue:Integer
  operations
    effect()
end

class RestorePreviousShoppingCart < DomainEvent, ExistingCustomerEvent
  operations
    effect()
end
association restorePreviousShoppingCart_session between
  RestorePreviousShoppingCart[*]
  Session[0..1]
end

class ReviewTextMinimumChange < DomainEvent
attributes
  newMinimum:Integer
operations
  effect()
end

class SendExtraOrderEmailChange < DomainEvent, StoreEvent
operations
  effect()
end

association sendExtraOrderEmailChange_newSendExtraOrderEmail between
  SendExtraOrderEmailChange[*]
  NameEMail[*] role newSendExtraOrderEMail
end

class SendNewsletter < DomainEvent, ExistingNewsletterEvent
operations
  effect()
end

class SetCancelledOrderStatus < DomainEvent, StoreEvent
operations
  effect()
end

association setCancelledOrderStatus_orderStatus between
  SetCancelledOrderStatus[*]
  OrderStatus[1]
end

class SetCurrentCurrency < DomainEvent, ExistingSessionEvent
operations
  effect()
end

association setCurrentCurrency_currency between
  SetCurrentCurrency[*]
  Currency[1] role newCurrentCurrency
end

class SetCurrentLanguage < DomainEvent, ExistingSessionEvent
operations
  effect()
end

association setCurrentLanguage_language between
  SetCurrentLanguage[*]
  Language[1] role newCurrentLanguage
end

class SetDefaultCurrency < DomainEvent, ExistingCurrencyEvent
operations
  effect()
end

class SetDefaultLanguage < DomainEvent, ExistingLanguageEvent
operations
  effect()
class SetDefaultOrderStatus < DomainEvent, StoreEvent
operations
  effect()
end

association SetDefaultOrderStatus_orderStatus between
  SetDefaultOrderStatus[*]
  OrderStatus[1]
end

class ShowBanner < DomainEvent, ExistingBannerEvent
operations
  effect()
end

class ShowBestPurchasedProducts < Query
operations
  answer():Set(Tuple(product:String, quantity:Integer)) =
  Product.allInstances
  -> sortedBy(quantityOrdered)
  -> collect (p | Tuple {product : ProductInLanguage.allInstances ->select
                     (pil | pil.product = p and
                      pil.language = language) -> any(true).name,
                     quantity : p.quantityOrdered}) -> asSet()
end

association showBestPurchasedProducts_language between
  ShowBestPurchasedProducts[*]
  Language[1]
end

class ShowBestViewedProducts < Query
operations
  answer():Set(Tuple(product:String, timesViewed:Integer)) =
  Product.allInstances
  -> sortedBy(timesViewed())
  -> collect (p | Tuple {product : ProductInLanguage.allInstances ->select
                        (pil | pil.product = p and
                         pil.language = language) -> any(true).name,
                        timesViewed : p.timesViewed()}) -> asSet()
end

association showBestViewedProducts_language between
  ShowBestViewedProducts[*]
  Language[1]
end

class ShowCustomersOrdersTotal < Query
operations
  answer():Set(Tuple(name:String, total:Real)) =
  Customer.allInstances
  -> collect (c | Tuple {name : c.firstName.concat(c.lastName),
                       total : c.order.total() -> sum()}) -> asSet()
end

class ShowExpectedProducts < Query
operations
  answer():Set(Tuple(product:String, dateAvailable:Date)) =
  Product.allInstances
  -> select(p | p.available.isDefined())
  -> sortedBy(dateAvailable)
  -> collect (p | Tuple {product : ProductInLanguage.allInstances ->select
                         (pil | pil.product = p and
                          pil.language = language) -> any(true).name,
                         dateAvailable : p.dateAvailable})->asSet()
class ShowNewProducts < Query
  operations
  answer(): Set(Tuple(product:String, added:DateTime))=
    Product.allInstances
    -> sortedBy(added.dateTime)
    -> collect (p | Tuple {product : ProductInLanguage.allInstances -> select
      (pil | pil.product = p and pil.language=language)->any(true).name,
      added : p.added}) -> asSet()
end

association showExpectedProducts_language between
  ShowExpectedProducts[*]
  Language[1]
end

class ShowNewProducts < Query
  operations
  answer(): Set(Tuple(product:String, added:DateTime))=
    Product.allInstances
    -> sortedBy(added.dateTime)
    -> collect (p | Tuple {product : ProductInLanguage.allInstances -> select
      (pil | pil.product = p and pil.language=language)->any(true).name,
      added : p.added}) -> asSet()
end

association showNewProducts_language between
  ShowNewProducts[*]
  Language[1]
end

class ShowOnlineCustomers < Query
  operations
  answer(): Set(String)=
    Session.allInstances.customer
    -> collect (c | c.firstName.concat(c.lastName)) -> asSet()
end

class ShowOrdersOfCustomer < Query, ExistingCustomerEvent
  operations
  answer(): Set(Tuple(id:Integer, total:Real, status:OrderStatus))=
    self.customer.order
    -> collect (o | Tuple {id : o.id(),
      total : o.total(),
      status : o.orderStatusChange-> last().orderStatus})
    -> asSet()
end

class ShowProductsOfCategory < Query, ExistingCategoryEvent
  operations
  answer(): Set(String)=
    Product.allInstances -> select(p | p.category -> includes(self.category))
    -> collect (p | ProductInLanguage.allInstances -> select
      (pil | pil.product = p and pil.language=language)->any(true).name) -> asSet()
end

association showProductsOfCategory_language between
  ShowProductsOfCategory[*]
  Language[1]
end

class ShowProductsOfManufacturer < Query, ExistingManufacturerEvent
  operations
  answer(): Set(String)=
    Product.allInstances -> select(p | p.manufacturer = self.manufacturer)
    -> collect (p | ProductInLanguage.allInstances -> select
      (pil | pil.product = p and pil.language=language)->any(true).name) -> asSet()
class ShowReviewsOfProduct < Query, ExistingProductEvent
operations
   answer(): Set(Tuple(review:String,rating:Rating))=
      self.product.review -> select (r | r.language = self.language)
         -> collect (r | Tuple {review : r.review,
         rating : r.rating})->asSet()
end
association showReviewsOfProduct_language between
   ShowReviewsOfProduct[*]
   Language[1]
end

class ShowSpecials < Query
operations
   answer(): Set(Tuple(product:String,oldPrice:Real, newPrice:Real))=
      Special.allInstances
         -> collect (s | Tuple {product : ProductInLanguage.allInstances ->select
         (pil | pil.product = s  and
         pil.language=language)->any(true).name,
         oldPrice : s.netPrice,
         newPrice : s.specialPrice})->asSet()
end
association showSpecials_language between
   ShowSpecials[*]
   Language[1]
end

class ShowUnderStockProducts < Query
operations
   answer(): Set(Tuple(product:String,quantity:Integer))=
      Product.allInstances -> select(p | p.quantityOnHand < Stock.allInstances
         ->any(true).stockReOrderLevel)
         -> collect (p | Tuple {product : ProductInLanguage.allInstances ->select
         (pil | pil.product = p and pil.language=language)->any(true).name,
         quantity : p.quantityOnHand}) -> asSet()
end
association showUnderStockProducts_language between
   ShowUnderStockProducts[*]
   Language[1]
end

class StateCustomerDetailChange < DomainEvent
attributes
   newValue:Boolean
operations
   effect()
end

class StateMinimumChange < DomainEvent
attributes
   newMinimum:Integer
operations
effect()
end
class StatusPaymentMethodChange < DomainEvent, ExistingPaymentMethodEvent
attributes
  newStatus:Status
operations
  effect()
end
class StatusShippingMethodChange < DomainEvent, ExistingShippingMethodEvent
attributes
  newStatus:Status
operations
  effect()
end
abstract class ExistingPaymentMethodEvent
end
association existingPaymentMethodEvent_paymentMethod between
  ExistingPaymentMethodEvent[*]
  PaymentMethod[1]
end
abstract class ExistingShippingMethodEvent
end
association existingShippingMethodEvent_shippingMethod between
  ExistingShippingMethodEvent[*]
  ShippingMethod[1]
end
class StoreAddressAndPhoneChange < DomainEvent, StoreEvent
attributes
  newStoreAddressAndPhone:String
operations
  effect()
end
class StreetAddressMinimumChange < DomainEvent
attributes
  newMinimum:Integer
operations
  effect()
end
class SubstractStockConfigurationChange < DomainEvent
attributes
  newValue:Boolean
operations
  effect()
end
class SuburbCustomerDetailChange < DomainEvent
attributes
  newValue:Boolean
operations
  effect()
end
class SwitchToDefaultLanguageCurrencyChange < DomainEvent, StoreEvent
attributes
  newSwitchToDefaultLanguageCurrency:Boolean
operations
  effect()
end
class TaxDecimalPlacesChange < DomainEvent, StoreEvent
attributes
  newTaxDecimalPlaces:Integer
operations
  effect()
end

class TelephoneMinimumChange < DomainEvent
attributes
  newMinimum:Integer
operations
  effect()
end

class TypicalPackageTareWeightShippingConfigurationChange < DomainEvent
attributes
  newValue:Integer
operations
  effect()
end

class UninstallAuthorizeNetPaymentMethod < DomainEvent
operations
  effect()
end

class UninstallCashOnDeliveryPaymentMethod < DomainEvent
operations
  effect()
end

class UninstallCheckMoneyPaymentMethod < DomainEvent
operations
  effect()
end

class UninstallCreditCardPaymentMethod < DomainEvent
operations
  effect()
end

class UninstallFlatRateShippingMethod < DomainEvent
operations
  effect()
end

class UninstallIPaymentPaymentMethod < DomainEvent
operations
  effect()
end

class UninstallNochexPaymentMethod < DomainEvent
operations
  effect()
end

class UninstallPayPalPaymentMethod < DomainEvent
operations
  effect()
end

class UninstallPerItemShippingMethod < DomainEvent
operations
  effect()
end
class UninstallPSiGatePaymentMethod < DomainEvent
  operations
   effect()
end

class UninstallSECPaymentMethod < DomainEvent
  operations
   effect()
end

class UninstallTableRateShippingMethod < DomainEvent
  operations
   effect()
end

class UninstallTwoCheckOutPaymentMethod < DomainEvent
  operations
   effect()
end

class UninstallUSPostalServiceShippingMethod < DomainEvent
  operations
   effect()
end

class UninstallZoneRatesShippingMethod < DomainEvent
  operations
   effect()
end

class UnlockNewsletter < DomainEvent, ExistingNewsletterEvent
  operations
   effect()
end

class UpdateCurrencyValueChange < DomainEvent, ExistingCurrencyEvent
  attributes
   newValue:Real
  operations
   effect()
end

class UpdateOrderStatus < DomainEvent, ExistingOrderEvent
  attributes
   comments:String
  operations
   effect()
end

association updateOrderStatus_zone between
  UpdateOrderStatus[*]
  OrderStatus[1] role newOrderStatus
end

class ZoneChange < DomainEvent, StoreEvent
  operations
   effect()
end

association zoneChange_zone between
  ZoneChange[*]
  Zone[1] role newZone
end

class UpdateShoppingCart < SessionEvent, ActionRequest
  operations
   effect()
end
abstract class ExistingShoppingCartItemEvent
end

association existingShoppingCartItemEvent_shoppingCartItem between
    ExistingShoppingCartItemEvent[*]
    ShoppingCartItem[1]
end

class LineChange
attributes
    index:Integer
    remove:Boolean
    quantity:Integer
end

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

class RemoveProduct < ExistingShoppingCartItemEvent
operations
    effect()
end

class ChangeQuantity < ExistingShoppingCartItemEvent
attributes
    quantity:Integer
operations
    effect()
end

-- CONSTRAINTS

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

context Store inv zoneIsPartOfCountry:
    self.zone->notEmpty() implies self.country.zone->includes(self.zone)

context ShippingAndPackaging inv tareIsLessThanMaximumWeight:
    self.typicalPackageTareWeight < self.maximumPackageWeight

context PaymentMethod inv atLeastOneEnabled:
    PaymentMethod.allInstances
        -> select(pm | pm.status=#enabled)->size() >= 1

context ShippingMethod inv atLeastOneEnabled:
    ShippingMethod.allInstances
        -> select(sm | sm.status=#enabled) -> size() >= 1

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

context Currency inv codeAndTitleAreUnique:
    Currency.allInstances->isUnique(title) and
    Currency.allInstances->isUnique(code)

context Country inv nameAndCodesAreUnique:
    Country.allInstances->isUnique(name) and
    Country.allInstances->isUnique(isoCode2) and
    Country.allInstances->isUnique(isoCode3)
context Zone inv nameAndCountryAndCodeAndCountryAreUnique:
  Zone.allInstances->isUnique(Tuple{n:name, c:country}) and
  Zone.allInstances->isUnique(Tuple{n:code, c:country})

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

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

context Language inv nameIsUnique:
  Language.allInstances->forAll(l |
    1.productInLanguage->isUnique(name))

context Language inv opcionNameIsUnique:
  self.hasOptionName->isUnique(optionName.string)

context Language inv valueNameIsUnique:
  self.hasValueName->isUnique(valueName.string)

context Language inv categoryNameIsUnique:
  self.hasCategoryName->isUnique(categoryName.string)

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

context Manufacturer inv nameIsUnique:
  Manufacturer.allInstances->isUnique(name)

context Manufacturer inv aURLInEachLanguage:
  self.language->size()=Language.allInstances->size()

context Banner inv titleIsUnique:
  Banner.allInstances->isUnique(title)

context BannerGroup inv nameIsUnique:
  BannerGroup.allInstances->isUnique(name)

context Newsletter inv titleIsUnique:
  Newsletter.allInstances->isUnique(title)

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

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

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

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

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

context Order inv ApplicableZoneShippingMethod:
  self.shippingMethod.oclIsTypeOf(SpecificZoneMethod) and
  self.shippingMethod.oclAsType(SpecificZoneMethod).taxZone -> notEmpty implies
  self.shippingMethod.oclAsType(SpecificZoneMethod).taxZone.zone
  -> includes(self.delivery.zone)

context Order inv ApplicableZoneRatesShippingMethod:
self.shippingMethod.oclIsTypeOf(ZoneRates) implies self.shippingMethod.oclAsType(ZoneRates).country -> includes(self.delivery.country)

context Order inv ApplicableZonesPaymentMethod:
    self.paymentMethod.taxZone -> notEmpty() implies self.paymentMethod.taxZone.zone -> includes(self.billing.zone)

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

custom OrderStatus inv NameIsUnique:
    Language.allInstances->forAll(
        l | l.orderStatus->isUnique(orderStatusInLanguage.name)
    )

context OrderStatus inv NameIsUnique:
    Language.allInstances->forAll(
        l | l.orderStatus->isUnique(orderStatusInLanguage.name)
    )

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

custom OrderStatus inv NameIsUnique:
    Language.allInstances->forAll(
        l | l.orderStatus->isUnique(orderStatusInLanguage.name)
    )

context Event CONSTRAINTS

context TypicalPackageTareWeightShippingConfigurationChange inv _iniIC_valueDoesNotExceedMaxWeight:
    self.newValue < ShippingAndPackaging.allInstances->any(true).maximumPackageWeight

context MaximumPackageWeightShippingConfigurationChange inv _iniIC_maxIsGreaterThanTypicalWeight:
    self.newMaximum > ShippingAndPackaging.allInstances
    ->any(true).typicalPackageTareWeight

context EditCreditCardPaymentMethod inv _iniIC_DoNotImpliesAllPaymentMethodsDisabled:
    PaymentMethod.allInstances -> select(pm | not(pm.oclIsTypeOf(CreditCard)))
    -> exists(pm | pm.status=#enabled)

context EditManufacturer inv _iniIC_manufacturerDoesNotExist:
    (Manufacturer.allInstances - Set{self.manufacturer}).name->excludes(self.name)

context NewCategory inv _iniIC_categoryDoesNotExist:
    Language.allInstances->forAll(l|
        l.hasCategoryName.categoryName.string->excludes(self.hasNewName
        ->select(languageOfCategory=l)->any(true).name.string)
    )

context EditCountry inv _iniIC_countryDoesNotExist:
    (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 EditZone inv _iniIC_zoneDoesNotExist:
    (Zone.allInstances - Set{self.zone}).name->excludes(self.newName) and
    (Zone.allInstances - Set{self.zone}).code->excludes(self.newCode)

context EditTaxClass inv _iniIC_taxClassDoesNotExist:
    (TaxClass.allInstances - Set{self.taxClass}).name->excludes(self.newName)

context EditTaxZone inv _iniIC_taxZoneDoesNotExist:
    (TaxZone.allInstances - Set{self.taxZone}).name->excludes(self.newName)

context EditBannerGroup inv _iniIC_bannerGroupDoesNotExist:
    (BannerGroup.allInstances - Set{self.bannerGroup}).name->excludes(self.newName)

context EditBanner inv _iniIC_bannerDoesNotExist:
    (Banner.allInstances - Set{self.banner}).title->excludes(self.newTitle)

context LockNewsletter inv _iniIC_newsletterIsNotLocked:
    self.newsletter.status <> #locked

context UnlockNewsletter inv _iniIC_newsletterIsNotUnlocked:
    self.newsletter.status <> #unlocked
context EditNewsletter inv _iniIC_newsletterIsUnlocked:
    self.newsletter.status = #unlocked

context EditNewsletter inv _iniIC_newsletterDoesNotExist:
    (Newsletter.allInstances - Set{self.newsletter}).title->excludes(self.newTitle)

context DeleteNewsletter inv _iniIC_newsletterIsUnlocked:
    self.newsletter.status = #unlocked

context EditTaxRate inv _iniIC_taxRateDoesNotExist:
    (TaxRate.allInstances - Set{self.taxRate})->select(tr |
        tr.taxClass = self.newTaxClass and tr.taxZone = self.newTaxZone)->size()=0

context EditPerItemShippingMethod inv _iniIC_DoNotImpliesAllShippingMethodsDisabled:
    ShippingMethod.allInstances -> select(sm | not(sm.oclIsTypeOf(PerItem)))
    -> exists(sm | sm.status=#enabled)

context AttributeChange inv _iniIC_OptionAndValueAreAValidAttribute:
    Attribute.allInstances->exists(a| a.option=self.newOption and a.value=newValue)

context MoveProduct inv _iniIC_oldCategoryIsValid:
    product.category->includes(self.oldCategory)

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

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

context DeleteBannerGroup inv _iniIC_BannerGroupIsEmpty:
    self.bannerGroup.banner -> isEmpty()

context DeleteCountry inv _iniIC_CountryIsNotALocation:
    Store.allInstances -> any(true).country <> self.country and Address.allInstances.country -> excludes(self.country)

context DeleteCurrency inv _iniIC_ExistsAnotherCurrencyEnabled:
    Currency.allInstances -> select (c| c<>self.currency) -> exists(c|c.status=#enabled)

context DeleteCustomerAddress inv _iniIC_AddressOfCustomer:
    self.customer.address -> includes(self.address)

context DeleteCustomerAddress inv _iniIC_AtLeastTwoAddresses:
    self.customer.address->size() >= 2

context DeleteCustomerAddress inv _iniIC_PrimaryAddressCannotBeDeleted:
    self.address <> self.customer.primary

context DeleteLanguage inv _iniIC_AtLeastTwoLanguages:
    Language.allInstances -> size() >= 2

context DeleteOrderStatus inv _iniIC_IsNotTheCurrentStatusOfAnyOrder:
    Order.allInstances -> forAll (o | o.orderStatusChange -> last().orderStatus <> self.orderStatus)

context DeleteOrderStatus inv _iniIC_IsNotADefaultStatus:
    Store.allInstances->forAll(s|
        s.defaultStatus <> self.orderStatus and s.cancelledStatus <> self.orderStatus)

context DeleteProductOption inv _iniIC_HasNoProductsOrValues:
    self.option.attribute.product -> isEmpty()
context DeleteProductOptionValue inv _iniIC_HasNotProducts:
    self.value.attribute.product -> isEmpty() and
    self.value.attribute.orderLineAttribute->isEmpty()

context DeleteZone inv _iniIC_ZoneIsNotALocation:
    Store.allInstances -> any(true).zone <> self.zone and
    Address.allInstances.zone -> excludes(self.zone)

context EditAuthorizeNetPaymentMethod inv _iniIC_PaymentMethodIsInstalled:
    AuthorizeNet.allInstances -> notEmpty()

context EditCashOnDeliveryPaymentMethod inv _iniIC_PaymentMethodIsInstalled:
    CashOnDelivery.allInstances -> notEmpty()

context EditCheckMoneyPaymentMethod inv _iniIC_PaymentMethodIsInstalled:
    CheckMoney.allInstances -> notEmpty()

context EditCreditCardPaymentMethod inv _iniIC_PaymentMethodIsInstalled:
    CreditCard.allInstances -> notEmpty()

context EditPerItemShippingMethod inv _iniIC_atLeastOneEnabled:
    self.status=#disabled implies
    (ShippingMethod.allInstances-Set{PerItem.allInstances->any(true)})->exists(pm | pm.status=#enabled)

context EditCustomer inv _iniIC_firstNameRight:
    self.newFirstName.size() >= MinimumValues.allInstances->any(true).firstName

context EditCustomer inv _iniIC_lastNameRight:
    self.newLastName.size() >= MinimumValues.allInstances->any(true).lastName

context EditCustomer inv _iniIC_dateOfBirthRight:
    CustomerDetails.allInstances->any(true).dateOfBirth implies
    self.newDateOfBirth.isDefined and
    self.newDateOfBirth.date.size() >= MinimumValues.allInstances->any(true).dateOfBirth

context EditCustomer inv _iniIC_genderRight:
    CustomerDetails.allInstances->any(true).gender implies self.newGender.isDefined()

context EditCustomer inv _iniIC_eMailRight:
    self.newEMailAddress.eMail.size() >= MinimumValues.allInstances->any(true).eMailAddress

context EditCustomer inv _iniIC_telephoneRight:
    self.newPhone.size() >= MinimumValues.allInstances->any(true).telephoneNumber

context EditLanguage inv _iniIC_languageDoesNotExist:
    not ((Language.allInstances-Set{self.language})->exists(name=self.newName or code=self.newCode))

context EditCurrency inv _iniIC_currencyDoesNotExist:
    not ((Currency.allInstances-Set{self.currency})->exists(title=self.newTitle or code=self.newCode))

context CurrencyStatusChange inv _iniIC_atLeastOneCurrencyEnabled:
    self.newStatus=#disabled implies
    (Currency.allInstances-Set{self.currency})->exists(c|c.status=#enabled)

context EditCustomerAddress inv _iniIC_AddressOfCustomer:
    self.customer.address -> includes(self.address)

context EditCustomerAddress inv _iniIC_firstNameRight:
    self.newAddress.firstName.size() >= MinimumValues.allInstances->any(true).firstName
context EditCustomerAddress inv _iniIC_lastNameRight:
    self.newAddress.lastName.size() >= MinimumValues.allInstances->any(true).lastName
context EditCustomerAddress inv _iniIC_genderRight:
    CustomerDetails.allInstances->any(true).gender implies self.
    newAddress.gender.isDefined()
context EditCustomerAddress inv _iniIC_suburbRight:
    CustomerDetails.allInstances->any(true).suburb implies self.
    newAddress.suburb.isDefined()
context EditCustomerAddress inv _iniIC_streetAddressRight:
    self.newAddress.street.size() >= MinimumValues.allInstances->any(true).streetAddress
context EditCustomerAddress inv _iniIC_companyRight:
    CustomerDetails.allInstances->any(true).company implies
    self.newAddress.company.isDefined() and
    self.newAddress.company.size() >= MinimumValues.allInstances
    ->any(true).companyName
context EditCustomerAddress inv _iniIC_postCodeRight:
    self.newAddress.postCode.postalCode.size() >= MinimumValues.allInstances
    ->any(true).postCode
context EditCustomerAddress inv _iniIC_cityRight:
    self.newAddress.city.size() >= MinimumValues.allInstances->any(true).city
context EditCustomerAddress inv _iniIC_stateRight:
    CustomerDetails.allInstances->any(true).state implies
    self.newAddress.state.isDefined() and
    self.newAddress.state.size() >= MinimumValues.allInstances->any(true).state
context EditCustomerAddress inv _iniIC_addressesHaveZoneIfNeeded:
    self.newAddress.zone->size()>0 implies
    self.newAddress.state = self.newAddress.zone.name and
    self.newAddress.country = self.newAddress.zone.country
context EditCustomerDetails inv _iniIC_firstNameRight:
    self.newFirstName.size() >= MinimumValues.allInstances->any(true).firstName
context EditCustomerDetails inv _iniIC_lastNameRight:
    self.newLastName.size() >= MinimumValues.allInstances->any(true).lastName
context EditCustomerDetails inv _iniIC_dateOfBirthRight:
    CustomerDetails.allInstances->any(true).dateOfBirth implies
    self.newDateOfBirth.isDefined() and
    self.newDateOfBirth.date.size() >= MinimumValues.allInstances
    ->any(true).dateOfBirth
context EditCustomerDetails inv _iniIC_genderRight:
    CustomerDetails.allInstances->any(true).gender implies self.newGender.isDefined()
context EditCustomerDetails inv _iniIC_eMailRight:
    self.newEMailAddress.eMail.size() >= MinimumValues.allInstances
    ->any(true).eMailAddress
context EditCustomerDetails inv _iniIC_telephoneRight:
    self.newPhone.size() >= MinimumValues.allInstances->any(true).telephoneNumber
context EditFlatRateShippingMethod inv _iniIC_PaymentMethodIsInstalled:
    FlatRate.allInstances -> notEmpty()
context EditIPaymentPaymentMethod inv _iniIC_PaymentMethodIsInstalled:
    IPayment.allInstances -> notEmpty()
context EditPerItemShippingMethod inv _iniIC_PaymentMethodIsInstalled:
   PerItem.allInstances -> notEmpty()

context EditPSiGatePaymentMethod inv _iniIC_PaymentMethodIsInstalled:
   PSiGate.allInstances -> notEmpty()

context EditSECPaymentMethod inv _iniIC_PaymentMethodIsInstalled:
   SECPay.allInstances -> notEmpty()

context EditTableRateShippingMethod inv _iniIC_PaymentMethodIsInstalled:
   TableRate.allInstances -> notEmpty()

context EditTwoCheckOutPaymentMethod inv _iniIC_PaymentMethodIsInstalled:
   TwoCheckOut.allInstances -> notEmpty()

context EditUSPostalServiceShippingMethod inv _iniIC_PaymentMethodIsInstalled:
   USPostalService.allInstances -> notEmpty()

context EditZoneRatesShippingMethod inv _iniIC_PaymentMethodIsInstalled:
   ZoneRates.allInstances -> notEmpty()

context InstallAuthorizeNetPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   AuthorizeNet.allInstances -> isEmpty()

context InstallCashOnDeliveryPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   CashOnDelivery.allInstances -> isEmpty()

context InstallCheckMoneyPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   AuthorizeNet.allInstances -> isEmpty()

context InstallCreditCardPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   CreditCard.allInstances -> isEmpty()

context InstallFlatRateShippingMethod inv _iniIC_ShippingMethodIsNotInstalled:
   FlatRate.allInstances -> isEmpty()

context InstallIPaymentPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   IPayment.allInstances -> isEmpty()

context InstallNochexPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   Nochex.allInstances -> isEmpty()

context InstallPayPalPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   PayPal.allInstances -> isEmpty()

context InstallPerItemShippingMethod inv _iniIC_ShippingMethodIsNotInstalled:
   PerItem.allInstances -> isEmpty()

context InstallPSiGatePaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   PSiGate.allInstances -> isEmpty()

context InstallSECPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
   SECPay.allInstances -> isEmpty()
context InstallTableRateShippingMethod inv _iniIC_ShippingMethodIsNotInstalled:
    TableRate.allInstances -> isEmpty()

context InstallTwoCheckOutPaymentMethod inv _iniIC_PaymentMethodIsNotInstalled:
    TwoCheckOut.allInstances -> isEmpty()

context InstallUSPostalServiceShippingMethod inv _iniIC_ShippingMethodIsNotInstalled:
    USPostalService.allInstances -> isEmpty()

context InstallZoneRatesShippingMethod inv _iniIC_ShippingMethodIsNotInstalled:
    ZoneRates.allInstances -> isEmpty()

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

context LogOut inv _iniIC_CustomerIsLoggedIn:
    self.session.customer = self.customer

context NewBanner inv _iniIC_bannerDoesNotExist:
    not Banner.allInstances ->exists (b | b.title= self.title)

context NewBannerGroup inv _iniIC_bannerGroupDoesNotExist:
    not BannerGroup.allInstances ->exists (bg | bg.name= self.name)

context NewCountry inv _iniIC_countryDoesNotExist:
    not Country.allInstances -> exists(c | c.name=self.name and 
        c.isoCode2=self.isoCode2 and 
        c.isoCode3 = self.isoCode3)

context NewCurrency inv _iniIC_currencyDoesNotExist:
    not (Currency.allInstances -> exists(c | c.title=self.title and 
        c.code=self.code))

context NewCustomer inv _iniIC_passwordCorrect:
    password = passwordConfirmation

context NewCustomer inv _iniIC_firstNameRight:
    self.primary.firstName.size() >= MinimumValues.allInstances->any(true).firstName

context NewCustomer inv _iniIC_lastNameRight:
    self.primary.lastName.size() >= MinimumValues.allInstances->any(true).lastName

context NewCustomer inv _iniIC_dateOfBirthRight:
    CustomerDetails.allInstances->any(true).dateOfBirth implies 
    self.dateOfBirth.isDefined() and 
    self.dateOfBirth.date.size() >= MinimumValues.allInstances->any(true).dateOfBirth

context NewCustomer inv _iniIC_genderRight:
    CustomerDetails.allInstances->any(true).gender implies 
    self.primary.gender.isDefined()

context NewCustomer inv _iniIC_suburbRight:
    CustomerDetails.allInstances->any(true).suburb implies 
    self.primary.suburb.isDefined()

context NewCustomer inv _iniIC_eMailRight:
    self.eMailAddress.eMail.size() >= MinimumValues.allInstances->any(true).eMailAddress

context NewCustomer inv _iniIC_streetAddressRight:
    self.primary.street.size() >= MinimumValues.allInstances->any(true).streetAddress

context NewCustomer inv _iniIC_companyRight:
    CustomerDetails.allInstances->any(true).company implies 
    self.primary.company.isDefined() and 
    self.primary.company.size() >= MinimumValues.allInstances->any(true).companyName
context NewCustomer inv _iniIC_postCodeRight:
  self.primary.postCode.postalCode.size() >= MinimumValues.allInstances
  ->any(true).postCode

context NewCustomer inv _iniIC_cityRight:
  self.primary.city.size() >= MinimumValues.allInstances->any(true).city

context NewCustomer inv _iniIC_stateRight:
  CustomerDetails.allInstances->any(true).state implies
  self.primary.state.isDefined() and
  self.primary.state.size() >= MinimumValues.allInstances->any(true).state

context NewCustomer inv _iniIC_telephoneRight:
  self.phone.size() >= MinimumValues.allInstances->any(true).telephoneNumber

context NewCustomer inv _iniIC_passwordRight:
  self.password.size() >= MinimumValues.allInstances->any(true).password

context NewCustomerAddress inv _iniIC_firstNameRight:
  self.firstName.size() >= MinimumValues.allInstances->any(true).firstName

context NewCustomerAddress inv _iniIC_lastNameRight:
  self.lastName.size() >= MinimumValues.allInstances->any(true).lastName

context NewCustomerAddress inv _iniIC_genderRight:
  CustomerDetails.allInstances->any(true).gender implies self.gender.isDefined()

context NewCustomerAddress inv _iniIC_suburbRight:
  CustomerDetails.allInstances->any(true).suburb implies self.suburb.isDefined()

context NewCustomerAddress inv _iniIC_streetAddressRight:
  self.street.size() >= MinimumValues.allInstances->any(true).streetAddress

context NewCustomerAddress inv _iniIC_companyRight:
  CustomerDetails.allInstances->any(true).company implies
  self.company.isDefined() and
  self.company.size() >= MinimumValues.allInstances->any(true).companyName

context NewCustomerAddress inv _iniIC_postCodeRight:

context NewCustomerAddress inv _iniIC_cityRight:
  self.city.size() >= MinimumValues.allInstances->any(true).city

context NewCustomerAddress inv _iniIC_stateRight:
  CustomerDetails.allInstances->any(true).state implies
  self.state.isDefined() and
  self.state.size() >= MinimumValues.allInstances->any(true).state

context NewCustomerAddress inv _iniIC_addressesHaveZoneIfNeeded:
  self.country.zone->size()>0 implies
  (self.state = self.zone.name and
   self.country = self.zone.country)

context NewCustomerAddress inv _iniIC_numberOfAddressesRight:
  self.customer.address -> size() < MaximumValues.allInstances
  ->any(true).addressBookEntries

context NewDownloadableProductAttribute inv _iniIC_productAttributeDoesNotExist:
  not ProductAttribute.allInstances -> exists (pa | pa.attribute.option =
  self.option and pa.attribute.value = self.value and
  pa.product = self.product)
context NewLanguage inv _iniIC_languageDoesNotExist:
not {Language.allInstances -> exists (l | l.name=self.newName and l.code = self.newCode)}

context NewManufacturer inv _iniIC_manufacturerDoesNotExist:
not Manufacturer.allInstances -> exists (m | m.name=self.name)

context NewNewsletter inv _iniIC_newsletterDoesNotExist:
not Newsletter.allInstances -> exists (n | n.title=self.title)

context NewOrderStatus inv _iniIC_orderStatusDoesNotExist:
not OrderStatus.allInstances -> exists (os | Language.allInstances->
exists(l |
  self.hasOrderStatusName
  ->select(languageOfOrderStatus=l).orderStatusName.string= os.orderStatusInLanguage->
  select(language=l).name))

context NewProduct inv _iniIC_productDoesNotExist:
Language.allInstances -> forAll ( l |
  l.productInLanguage.name
  -> excludes(self.hasNewProductName
  -> any(languageOfProduct=l).nameOfProduct.string))

context EditProduct inv _iniIC_productDoesNotExist:
Language.allInstances -> forAll ( l |
  l.productInLanguage.name
  -> excludes(self.hasNewProductName
  -> any(languageOfProduct=l).nameOfProduct.string) or
  (self.hasNewProductName->any(languageOfProduct=l).nameOfProduct.string =
  self.product.productInLanguage->any(language=l).name))

context EditProductOption inv _iniIC_productOptionDoesNotExist:
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 EditCategory inv _iniIC_categoryDoesNotExist:
Language.allInstances -> forAll ( l |
  l.hasCategoryName.categoryName.string
  -> excludes(self.hasNewName -> any(languageOfCategory=l).name.string) or
  (self.hasNewName->any(languageOfCategory=l).name.string =
  self.category.hasCategoryName->any(language=l).categoryName.string))

context EditOrderStatus inv _iniIC_orderStatusDoesNotExist:
Language.allInstances -> forAll ( l |
  l.orderStatusInLanguage.name
  ->excludes(self.hasOrderStatusName
  -> any(languageOfOrderStatus=l).orderStatusName.string)
  or
  l.orderStatusInLanguage->any(orderStatus=self.orderStatus).name =
  self.hasOrderStatusName->any(languageOfOrderStatus=l).orderStatusName.string)

context EditCategory inv _iniIC_cyclesDoNotAppear:
self.category.allParents()->union(Set{self.newParent})->excludes(self.category)

context MoveCategory inv _iniIC_cyclesDoNotAppear:
self.newParent.allParents() ->excludes(self.category)

context EditProductOptionValue inv _iniIC_productOptionValueDoesNotExist:
Language.allInstances -> forAll ( l |
  l.hasValueName.valueName
  -> excludes(self.hasNewValueName -> any(languageOfValue=l).nameOfValue) or
context NewProductAttribute inv _iniIC_productAttributeDoesNotExist:
  not self.product.productAttribute
  -> exists(attribute.value=self.value and
            attribute.option = self.option)

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

context NewProductNotification inv _iniIC_ProductNotificationDoesNotExist:
  not Newsletter.allInstances -> exists (n | n.title = self.title)

context NewProductNotificationSubscription inv _iniIC_ProductIsUnsubscribed:
  not self.customer.globalNotifications and
  self.customer.explicitNotifications -> excludes(self.newSubscribedProduct)

context NewProductOption inv _iniIC_productOptionDoesNotExist:
  Language.allInstances -> forAll ( l |
        l.hasOptionName.optionName
        -> excludes(self.hasNewOptionName
                -> select(languageOfOption=l).nameOfOption->any(true)))

context NewProductOptionValue inv _iniIC_optionValueDoesNotExist:
  Language.allInstances -> forAll ( l |
        l.hasValueName.valueName.string
        -> excludes(self.hasNewValueName
                -> select(languageOfValue=l).nameOfValue
                ->any(true).string))

context NewReview inv _iniIC_reviewRight:
  self.review.size() >= MinimumValues.allInstances->any(true).reviewText

context NewTaxClass inv _iniIC_TaxClassDoesNotExist:
  not TaxClass.allInstances -> exists (tc | tc.name = self.name)

context NewTaxRate inv _iniIC_TaxRateDoesNotExist:
  not TaxRate.allInstances -> exists (tr | tr.taxClass = self.taxClass and
                                     tr.taxZone = self.taxZone)

context NewTaxZone inv _iniIC_TaxZoneDoesNotExist:
  not TaxZone.allInstances -> exists (tz | tz.name = self.name)

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

context OrderConfirmation inv _iniIC_PaymentMethodIsEnabled:
  self.paymentMethod.status= #enabled

context OrderConfirmation inv _iniIC_CurrencyIsEnabled:
  self.currency.status = #enabled

context OrderConfirmation inv _iniIC_CreditCardDetailsNeeded:
  self.paymentMethod.oclIsTypeOf(AuthorizeNet) or
  self.paymentMethod.oclIsTypeOf(CreditCard) or
  self.paymentMethod.oclIsTypeOf(IPayment) or
  self.paymentMethod.oclIsTypeOf(TwoCheckOut) or
  self.paymentMethod.oclIsTypeOf(PSiGate)
creditCardType.isDefined() and
creditCardOwner.isDefined() and
creditCardNumber.isDefined() and
creditCardExpires.isDefined()

context OrderConfirmation inv _iniIC_StockAllowsOrder:
    Stock.allInstances->any(true).allowCheckout or
    not Stock.allInstances->any(true).checkStockLevel or
    (self.shoppingCart.shoppingCartItem.product -> forAll (p | p.quantityOnHand > 0))

context PasswordChange inv _iniIC_passwordRight:
    self.newPassword.size() >= MinimumValues.allInstances->any(true).password

context PasswordChange inv _iniIC_oldPasswordIsCorrect:
    self.customer.password = self.oldPassword

context PrimaryCustomerAddressChange inv _iniIC_AddressOfCustomer:
    self.customer.address -> includes(self.address)

context ProductDownload inv _iniIC_DownloadEnabled:
    Download.allInstances->any(true).enableDownload

context ProductDownload inv _iniIC_ProductWasPurchasedByCustomer:
    self.customer.order.orderLine.product -> includes(self.product)

context ProductDownload inv _iniIC_DownloadableIsFromProduct:
    self.product.productAttribute -> select(pa | pa.oclIsTypeOf(Downloadable))
    -> includes(self.downloadable)

context ProductDownload inv _iniIC_DownloadsCountNotExceeded:
    let DownloadCountFromProduct:Integer =
    self.customer.order.orderLine.orderLineAttribute
    -> select (ola | ola.oclIsTypeOf(OrderDownload) and
    ola.orderLine.product = self.product)
    ->asSequence()->last().oclAsType(OrderDownload).downloadCount
    in
    DownloadCountFromProduct < self.downloadable.maximumDownloadCount

context RestorePreviousShoppingCart inv _iniIC_CustomerHasAPreviousShoppingCart:
    self.customer.customerShoppingCart->notEmpty()

context UninstallAuthorizeNetPaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    AuthorizeNet.allInstances -> notEmpty()

context UninstallCashOnDeliveryPaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    CashOnDelivery.allInstances -> notEmpty()

context UninstallCheckMoneyPaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    AuthorizeNet.allInstances -> notEmpty()

context UninstallCreditCardPaymentMethod inv _iniIC_PaymentMethodCanBeUninstalled:
    CreditCard.allInstances -> notEmpty() and
    (PaymentMethod.allInstances-Set{CreditCard.allInstances->any(true)})
    ->exists(pm|pm.status=#enabled)

context UninstallFlatRateShippingMethod inv _iniIC_ShippingMethodIsNotUninstalled:
    FlatRate.allInstances -> notEmpty()

context UninstallIPaymentPaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    IPayment.allInstances -> notEmpty()

context UninstallNochexPaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    Nochex.allInstances -> notEmpty()
context UninstallPayPalPaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    PayPal.allInstances -> notEmpty()

context UninstallPerItemShippingMethod inv _iniIC_ShippingMethodCanBeUninstalled:
    PerItem.allInstances -> notEmpty() and
    (ShippingMethod.allInstances-Set{PerItem.allInstances->any(true)})
    ->exists(sm|sm.status=#enabled)

context UninstallPSiGatePaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    PSiGate.allInstances -> notEmpty()

context UninstallSECPaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    SECPay.allInstances -> notEmpty()

context UninstallTableRateShippingMethod inv _iniIC_ShippingMethodIsNotUninstalled:
    TableRate.allInstances -> notEmpty()

context UninstallTwoCheckOutPaymentMethod inv _iniIC_PaymentMethodIsNotUninstalled:
    TwoCheckOut.allInstances -> notEmpty()

context UninstallUSPostalServiceShippingMethod inv
    _iniIC_ShippingMethodIsNotUninstalled:
    USPostalService.allInstances -> notEmpty()

context UninstallZoneRatesShippingMethod inv _iniIC_ShippingMethodIsNotUninstalled:
    ZoneRates.allInstances -> notEmpty()

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

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

context AddressBookEntriesMaximumChange::effect()
post: MaximumValues.allInstances->any(true).addressBookEntries = self.newMaximum

context AllowCheckoutStockConfigurationChange::effect()
post: Stock.allInstances->any(true).allowCheckout = self.newValue

context AllowGuestToTellAFriendChange::effect()
post: myStore().allowGuestToTellAFriend = self.newAllowGuestToTellAFriend

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

context CancelOrder::effect()
post:
self.order.orderStatusChange -> last().orderStatus =
Store.allInstances ->any(true).cancelledStatus

context CheckLevelStockConfigurationChange::effect()
post: Stock.allInstances->any(true).checkStockLevel = self.newValue

context CityMinimumChange::effect()
post: MinimumValues.allInstances->any(true).city = self.newMinimum

context ClickBanner::effect()
post:
BannerHistory.allInstances -> one
(bh | bh.banner = self.banner and
bh.clicked = bh@pre.clicked + 1)

context ClickManufacturer::effect()
post:
let manufacturerLanguageRead:ManufacturerInLanguage =
ManufacturerInLanguage.allInstances -> select
(mil | mil.manufacturer = self.manufacturer and
mil.language = self.language) ->any(true)
in
manufacturerLanguageRead.urlClicked =
manufacturerLanguageRead@pre.urlClicked + 1

context CompanyCustomerDetailChange::effect()
post: CustomerDetails.allInstances->any(true).company = self.newValue

context CompanyNameMinimumChange::effect()
post: MinimumValues.allInstances->any(true).companyName = self.newMinimum

context CountryChange::effect()
post: myStore().country = self.newCountry
context CountryShippingConfigurationChange::effect()
  post : ShippingAndPackaging.allInstances->any(true).countryOfOrigin = self.newCountryOfOrigin

context CreditCardNumberMinimumChange::effect()
  post : MinimumValues.allInstances->any(true).creditCardNumber = self.newMinimum

context CreditCardOwnerNameMinimumChange::effect()
  post : MinimumValues.allInstances->any(true).creditCardOwnerName = self.newMinimum

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

context CustomerStatusChange::effect()
  post : self.customer.status = self.newStatus

context DateOfBirthCustomerDetailChange::effect()
  post : CustomerDetails.allInstances->any(true).dateOfBirth = self.newValue

context DateOfBirthMinimumChange::effect()
  post : MinimumValues.allInstances->any(true).dateOfBirth = self.newMinimum

context DaysExpiryDelayDownloadConfigurationChange::effect()
  post : Download.allInstances->any(true).daysExpiryDelay = self.newValue

context DefaultSearchOperatorChange::effect()
  post : myStore().defaultSearchOperator = self.newDefaultSearchOperator

context DeleteBanner::effect()
  post : Banner.allInstances->excludes(self.banner@pre)

context DeleteBannerGroup::effect()
  post : BannerGroup.allInstances->excludes(self.bannerGroup@pre)

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 -> forAll(p | p.orderLine->notEmpty() then p.status=#outOfStock else Product.allInstances->excludes(p@pre) endif)
  post deleteProductsOfChildCategories:
    self.allChilds(category@pre).product -> forAll(p | p.orderLine->notEmpty() then p.status=#outOfStock else Product.allInstances->excludes(p@pre) endif)

context DeleteCountry::effect()
  post : Country.allInstances->excludes(self.country@pre)
  post : self.country@pre.zone -> forAll(z | Zone.allInstances->excludes(z))

context DeleteCurrency::effect()
  post : Currency.allInstances->excludes(self.currency@pre)

context DeleteCustomer::effect()
  post deleteCustomer:
    if customer@pre.order->size()=0 then
      Customer.allInstances->excludes(customer@pre)
    else
      customer.status=#disabled
    endif
context DeleteCustomerAddress::effect()
  post: self.customer.address -> excludes(self.address)

custom DeleteLanguage::effect()
  post: not Language.allInstances->exists(l | l=self.language@pre)

custom DeleteManufacturer::effect()
  post deleteManufacturer:
    Manufacturer.allInstances->excludes(self.manufacturer@pre)
    changeProductsToOutOfStock:
      deleteProds implies
      manufacturer@pre.product@pre ->
      forAll(status = #outOfStock)

custom DeleteNewsletter::effect()
  post: Newsletter.allInstances->excludes(self.newsletter@pre)

custom DeleteOrderStatus::effect()
  post: if Order.allInstances.orderStatus->includes(self.orderStatus)
    then self.orderStatus.status=#disabled
    else OrderStatus.allInstances->excludes(self.orderStatus@pre)
  endif

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

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

custom DeleteProductOption::effect()
  post: Option.allInstances->excludes(self.option@pre)
  post: self.option@pre.value->select(v | {v.option-Set{self.option@pre}}->isEmpty() or
  v.attribute.orderLineAttribute->isEmpty())
  -> forAll(v | Value.allInstances->excludes(v))

custom DeleteProductOptionValue::effect()
  post: Value.allInstances->excludes(self.value@pre)

custom DeleteReview::effect()
  post: Review.allInstances->excludes(self.review@pre)
context DeleteSession::effect()
  post : Session.allInstances->excludes(self.session@pre)

context DeleteSpecial::effect()
  post :
  Special.allInstances->excludes(special@pre) and
  Product.allInstances - Product.allInstances@pre -> forAll(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.modelM = special@pre.modelM@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)
  Language.allInstances ->
  forAll (l |
  special@pre.productInLanguage->select(language=l).name = p.productInLanguage->select(language=l).name)

context DeleteTaxClass::effect()
  post deleteTaxClass:
  TaxClass.allInstances->excludes(self.taxClass@pre)
  post deleteAssociatedTaxRates:
  self.taxClass@pre.taxRate@pre -> forAll(tr | TaxRate.allInstances->excludes(tr))

context DeleteTaxRate::effect()
  post : TaxRate.allInstances->excludes(self.taxRate@pre)

context DeleteTaxZone::effect()
  post deleteTaxZone:
  TaxZone.allInstances->excludes(self.taxZone@pre)
  post deleteAssociatedTaxRates:
  self.taxZone@pre.taxRate@pre -> forAll(tr | TaxRate.allInstances->excludes(tr))

context DeleteZone::effect()
  post : Zone.allInstances->excludes(self.zone@pre)

context DisplayCartAfterAddingProductChange::effect()
  post : myStore().displayCartAfterAddingProduct = self.newDisplayCartAfterAddingProduct

context DisplayPricesWithTaxChange::effect()
  post : myStore().displayPricesWithTax = self.newDisplayPricesWithTax

context EditAuthorizeNetPaymentMethod::effect()
  post :
  let pm:AuthorizeNet = AuthorizeNet.allInstances -> any(true) in
  pm.username=self.newUsername and pm.key=self.newKey and pm.mode=self.newMode and pm.method=self.newMethod and pm.notification=self.newNotification and pm.orderStatus=self.orderStatus and pm.status=self.status and pm.taxZone=self.taxZone

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.newBannerGroup

color EditBannerGroup::effect()
post : self.bannerGroup.name = self.newName

color EditCashOnDeliveryPaymentMethod::effect()
post :
let pm: CashOnDelivery = CashOnDelivery.allInstances -> any(true) in
pm.orderStatus = self.orderStatus and
pm.status = self.status and
pm.taxZone = self.taxZone

color EditCategory::effect()
post :
self.category.imagePath = self.imagePath and
self.category.sortOrder = self.sortOrder and
self.category.parent = self.newParent and
Language.allInstances -> forAll (l |
  self.hasNewName -> select(languageOfCategory = l) -> any(true).name.string =
  self.category.hasCategoryName -> select(language = l).categoryName
  -> any(true).string )

color EditCheckMoneyPaymentMethod::effect()
post :
let pm: CheckMoney = CheckMoney.allInstances -> any(true) in
pm.makePayableTo = self.newMakePayableTo and
pm.orderStatus = self.orderStatus and
pm.status = self.status and
pm.taxZone = self.taxZone

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

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

color 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

color 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

class EditCustomerAddress::effect()

post :
  self.customer.address -> excludes(self.address) and
  self.customer.address -> includes(self.newAddress)

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

class EditDownloadableAttribute::effect()

post :
  self.downloadable.filename = self.newFilename and
  self.downloadable.expiryDays = self.newExpiryDays and
  self.downloadable.maximumDownloadCount = self.newMaximumDownloadCount

context EditFlatRateShippingMethod::effect()

post :
  let sm: FlatRate = FlatRate.allInstances -> any(true) in
  sm.cost = self.newCost and
  sm.taxZone = self.taxZone and
  sm.taxClass = self.taxClass and
  sm.status = self.status

context EditGlobalNotifications::effect()

post :
  self.customer.globalNotifications = self.newGlobalNotifications

context EditIPaymentPaymentMethod::effect()

post :
  let pm: IPayment = IPayment.allInstances -> any(true) in
  pm.account = self.newAccount and
  pm.user = self.newUser and
  pm.password = self.newPassword and
  pm.status = self.status and
  pm.orderStatus = self.orderStatus and
  pm.taxZone = self.taxZone

context EditLanguage::effect()

post :
  self.language.name = self.newName and
  self.language.code = self.newCode and
  self.language.defaultCurrency = self.newDefaultCurrency

context EditManufacturer::effect()

post :
  self.manufacturer.name = self.name and
  self.manufacturer.imagePath = self.imagePath and
  Language.allInstances -> forAll(l)
    self.hasURL -> select(languageOfURL=l).url=
      self.manufacturer.manufacturerInLanguage->
        select(language=l).url

context EditNewsletter::effect()

post :
  newsletter.title = self.newTitle and
newsletter.content = self.newContent

class EditNochexPaymentMethod::effect()
post:
  let pm: Nochex = Nochex.allInstances -> any(true) in
  pm.eMail = self.newEMail and
  pm.status = self.status and
  pm.orderStatus = self.orderStatus and
  pm.taxZone = self.taxZone

context EditOrderStatus::effect()
post:
  Language.allInstances ->
      forAll(l| self.hasOrderStatusName
                -> select(languageOfOrderStatus=l).orderStatusName.string =
                self.orderStatus.orderStatusInLanguage
                -> select(language=l).name)

context EditPayPalPaymentMethod::effect()
post:
  let pm: PayPal = PayPal.allInstances -> any(true) in
  pm.eMail = self.newEMail and
  pm.status = self.status and
  pm.orderStatus = self.orderStatus and
  pm.taxZone = self.taxZone

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

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.modelM = self.modelM 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(languageOfProduct=l).nameOfProduct
               -> any(true).string =
               self.product.productInLanguage->select(language=l).name
               -> any(true)
     )

context EditProductNotification::effect()
post:
  self.productNotification.global = self.newGlobal and
  self.productNotification.explicitNotifications = self.newExplicitNotifications

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

context EditProductOptionValue::effect()
post:
context EditPSiGatePaymentMethod::effect()
  post :
    let pm: PSiGate = PSiGate.allInstances -> any(true) in
    pm.merchantID = self.newMerchantID and
    pm.mode = self.newMode and
    pm.type = self.newType and
    pm.creditCardCollection = self.newCreditCardCollection and
    pm.status = self.status and
    pm.orderStatus = self.orderStatus and
    pm.taxZone = self.taxZone

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

context EditSECPaymentMethod::effect()
  post :
    let pm: SECPay = SECPay.allInstances -> any(true) in
    pm.merchantID = self.newMerchantID and
    pm.mode = self.newMode and
    pm.status = self.status and
    pm.orderStatus = self.orderStatus and
    pm.taxZone = self.taxZone

context EditSpecial::effect()
  post :
    self.special.specialPrice = self.newSpecialPrice and
    self.special.expiryDate = self.newExpiryDate and
    self.special.specialStatus = self.newStatus

context EditTableRateShippingMethod::effect()
  post :
    let sm: TableRate = TableRate.allInstances -> any(true) in
    sm.items = self.newItems and
    sm.method = self.newMethod and
    sm.handlingFee = self.handlingFee and
    sm.taxZone = self.taxZone and
    sm.taxClass = self.taxClass and
    sm.status = self.status

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

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

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

context EditTwoCheckOutPaymentMethod::effect()
post :
let pm: TwoCheckOut = TwoCheckOut.allInstances -> any(true) in
pm.login=self.newLogin and
pm.mode=self.newMode and
pm.merchantNotification=self.newMerchantNotification and
pm.status=self.status and
pm.orderStatus=self.orderStatus and
pm.taxZone=self.taxZone

context EditUSPostalServiceShippingMethod::effect()
post :
let sm: USPostalService= USPostalService.allInstances -> any(true) in
sm.userID=self.newUserID and
sm.password=self.newPassword and
sm.server=self.newServer and
sm.handlingFee=self.handlingFee and
sm.taxZone=self.taxZone and
sm.taxClass=self.taxClass and
sm.status = self.status

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

context EditZoneRatesShippingMethod::effect()
post :
let sm: ZoneRates= ZoneRates.allInstances -> any(true) in
sm.items=self.newItems and
sm.country=self.country and
sm.taxClass=self.taxClass and
sm.status=self.status

context EMailAddressChange::effect()
post :  myStore().eMailAddress = self.newEmailAddress

context EMailAddressMinimumChange::effect()
post :  MinimumValues.allInstances->any(true).eMailAddress = self.newMinimum

context EMailFromChange::effect()
post :  myStore().eMailFrom = self.newEmailFrom

context EnableDownloadConfigurationChange::effect()
post :  Download.allInstances->any(true).enableDownload= self.newValue

context ExpectedSortFieldChange::effect()
post :  myStore().expectedSortField = self.newExpectedSortField

context ExpectedSortOrderChange::effect()
post :  myStore().expectedSortOrder = self.newExpectedSortOrder

context FirstNameMinimumChange::effect()
post :  MinimumValues.allInstances->any(true).firstName = self.newMinimum

context GenderCustomerDetailChange::effect()
post :  CustomerDetails.allInstances->any(true).gender = self.newValue

context IncrementAndSignAttributeChange::effect()
post :  self.productAttribute.increment = self.newIncrement and
        self.productAttribute.sign = self.newSign
context InstallAuthorizeNetPaymentMethod::effect()
post : (AuthorizeNet.allInstances - AuthorizeNet.allInstances@pre)
   -> forAll(pm:AuthorizeNet | pm.oclIsNew() and pm.oclIsTypeOf(AuthorizeNet) and pm.status=#enabled)

context InstallCashOnDeliveryPaymentMethod::effect()
post : (CashOnDelivery.allInstances - CashOnDelivery.allInstances@pre) ->
   forAll(pm:CashOnDelivery | pm.oclIsNew() and pm.oclIsTypeOf(CashOnDelivery) and pm.status=#enabled)

context InstallCheckMoneyPaymentMethod::effect()
post : (CheckMoney.allInstances - CheckMoney.allInstances@pre) ->
   forAll(pm:CheckMoney | pm.oclIsNew() and pm.oclIsTypeOf(CheckMoney) and pm.status=#enabled)

context InstallCreditCardPaymentMethod::effect()
post : (CreditCard.allInstances - CreditCard.allInstances@pre) ->
   forAll(pm:CreditCard | pm.oclIsNew() and pm.oclIsTypeOf(CreditCard) and pm.status=#enabled)

context InstallFlatRateShippingMethod::effect()
post : (FlatRate.allInstances - FlatRate.allInstances@pre) ->
   forAll(sm:FlatRate | sm.oclIsNew() and sm.oclIsTypeOf(FlatRate) and sm.status=#enabled)

context InstallIPaymentPaymentMethod::effect()
post : (IPayment.allInstances - IPayment.allInstances@pre) ->
   forAll(pm:IPayment | pm.oclIsNew() and pm.oclIsTypeOf(IPayment) and pm.status=#enabled)

context InstallNochexPaymentMethod::effect()
post : (Nochex.allInstances - Nochex.allInstances@pre) ->
   forAll(pm:Nochex | pm.oclIsNew() and pm.oclIsTypeOf(Nochex) and pm.status=#enabled)

context InstallPayPalPaymentMethod::effect()
post : (PayPal.allInstances - PayPal.allInstances@pre) ->
   forAll(pm:PayPal | pm.oclIsNew() and pm.oclIsTypeOf(PayPal) and pm.status=#enabled)

context InstallPerItemShippingMethod::effect()
post : (PerItem.allInstances - PerItem.allInstances@pre) ->
   forAll(sm:PerItem | sm.oclIsNew() and sm.oclIsTypeOf(PerItem) and sm.status=#enabled)

context InstallPSiGatePaymentMethod::effect()
post : (PSiGate.allInstances - PSiGate.allInstances@pre) ->
   forAll(pm:PSiGate | pm.oclIsNew() and pm.oclIsTypeOf(PSiGate) and pm.status=#enabled)

context InstallSECPaymentMethod::effect()
post : (SECPay.allInstances - SECPay.allInstances@pre) ->
   forAll(pm:SECPay | pm.oclIsNew() and pm.oclIsTypeOf(SECPay) and pm.status=#enabled)

context InstallTableRateShippingMethod::effect()
post : (TableRate.allInstances - TableRate.allInstances@pre) ->
   forAll(sm:TableRate | sm.oclIsNew() and sm.oclIsTypeOf(TableRate) and sm.status=#enabled)

context InstallTwoCheckOutPaymentMethod::effect()
post : (TwoCheckOut.allInstances - TwoCheckOut.allInstances@pre) ->
   forAll(pm:TwoCheckOut | pm.oclIsNew() and pm.oclIsTypeOf(TwoCheckOut) and pm.status=#enabled)
context InstallUSPostalServiceShippingMethod::effect()
post : (USPostalService.allInstances - USPostalService.allInstances@pre)
-> forAll(sm:USPostalService | sm.oclIsNew() and sm.oclIsTypeOf(USPostalService) and sm.status=#enabled)

context InstallZoneRatesShippingMethod::effect()
post : (ZoneRates.allInstances - ZoneRates.allInstances@pre) -> forAll(sm:ZoneRates | sm.oclIsNew() and sm.oclIsTypeOf(ZoneRates) and sm.status=#enabled)

context LastNameMinimumChange::effect()
post : MinimumValues.allInstances->any(true).lastName = self.newMinimum

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

context LockNewsletter::effect()
post : self.newsletter.status = #locked

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 (RestorePreviousShoppingCart.allInstances - RestorePreviousShoppingCart.allInstances@pre)
-> forAll(rpsc:RestorePreviousShoppingCart | rpsc.oclIsNew() and rpsc.oclIsTypeOf(RestorePreviousShoppingCart) and rpsc.customer=self.customer and rpsc.session=self.session)
else
if self.session.shoppingCart->notEmpty() then (CustomerShoppingCart.allInstances - CustomerShoppingCart.allInstances@pre)
-> one(csc:CustomerShoppingCart | 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

context LogOut::effect()
post : self.session.customer -> isEmpty()

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

context MaximumNumberDownloadConfigurationChange::effect()
post : Download.allInstances->any(true).maximumNumberOfDownloads= self.newMaximum

context MaximumPackageWeightShippingConfigurationChange::effect()
post : ShippingAndPackaging.allInstances->any(true).maximumPackageWeight = self.newMaximum

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

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

context NewBanner::effect()
post :
  (Banner.allInstances - Banner.allInstances@pre) -> forAll (b:Banner |
  boclIsNew() and
  boclIsTypeOf(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 = #enabled)

context NewBannerGroup::effect()
post :
  (BannerGroup.allInstances - BannerGroup.allInstances@pre) -> forAll (bg:BannerGroup |
  bgoclIsNew() and
  bgoclIsTypeOf(BannerGroup) and
  bg.name = self.name)

context NewCategory::effect()
post :
  (Category.allInstances - Category.allInstances@pre) -> forAll (c:Category |
  coclIsNew() and
  coclIsTypeOf(Category) and
  c.imagePath = self.imagePath and
  c.sortOrder = self.sortOrder and
  c.parent = self.parent and
  Language.allInstances -> forAll (l| self.hasNewName -> select(languageOfCategory=l)->any(true).name =
  c.hasCategoryName->select(language=l)->any(true).categoryName))

context NewCountry::effect()
post :
  (Country.allInstances - Country.allInstances@pre) -> forAll (c:Country |
  coclIsNew() and
  coclIsTypeOf(Country) and
  c.name = self.name and
  c.isoCode2 = self.isoCode2 and
  c.isoCode3 = self.isoCode3
  )

context NewCurrency::effect()
post :
  (Currency.allInstances - Currency.allInstances@pre) -> forAll (c:Currency |
  coclIsNew() and
  coclIsTypeOf(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 = #enabled)

context NewCustomer::effect()
pre: not Customer.allInstances -> exists (c | c.eMailAddress = self.eMailAddress)
post :
  (Customer.allInstances - Customer.allInstances@pre) -> forAll (c:Customer |
  coclIsNew() and
  coclIsTypeOf(Customer) and
  c.gender = self.primary.gender and
  c.firstName = self.primary.firstName and
  c.lastName = self.primary.lastName and
  c.receivedEmail = self.receivedEmail and
  c.successEmail = self.successEmail and
  c.failEmail = self.failEmail and
  c.lang = self.lang
  )

context NewLanguage::effect()
post :
  (Language.allInstances - Language.allInstances@pre) -> forAll (l:Language |
  loclIsNew() and
  loclIsTypeOf(Language) and
  l.name = self.name
  )

context NewMenu::effect()
post :
  (Menu.allInstances - Menu.allInstances@pre) -> forAll (m:Menu |
  moclIsNew() and
  moclIsTypeOf(Menu) and
  m.name = self.name
  )

context NewPage::effect()
post :
  (Page.allInstances - Page.allInstances@pre) -> forAll (p:Page |
  poclIsNew() and
  poclIsTypeOf(Page) and
  p.name = self.name and
  p.address = self.address and
  p.enterUrl = self.enterUrl and
  p.address = self.address and
  p.address = self.address and
  p.address = self.address
  )

context NewProduct::effect()
post :
  (Product.allInstances - Product.allInstances@pre) -> forAll (p:Product |
  poclIsNew() and
  poclIsTypeOf(Product) and
  p.name = self.name and
  p.type = self.type and
  p.productCode = self.productCode and
  p.description = self.description and
  p.url = self.url and
  p.price = self.price and
  p.quantity = self.quantity and
  p.status = #enabled
  )

context NewRegion::effect()
post :
  (Region.allInstances - Region.allInstances@pre) -> forAll (r:Region |
  rocIsNew() and
  rocIsTypeOf(Region) and
  r.country = self.country and
  r.language = self.language and
  r.alternateLanguage = self.alternateLanguage
  )

context NewSupplier::effect()
post :
  (Supplier.allInstances - Supplier.allInstances@pre) -> forAll (s:Supplier |
  socIsNew() and
  socIsTypeOf(Supplier) and
  s.name = self.name and
  s.address = self.address and
  s.city = self.city and
  s.country = self.country and
  s.postalCode = self.postalCode and
  s.phoneNumber = self.phoneNumber and
  s.email = self.email and
  s.url = self.url
  )

context NewUser::effect()
post :
  (User.allInstances - User.allInstances@pre) -> forAll (u:User |
  uoclIsNew() and
  uoclIsTypeOf(User) and
  u.name = self.name and
  u.email = self.email and
  u.password = self.password and
  u.role = self.role and
  u.status = self.status
  )
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)

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))

context NewDownloadableProductAttribute::effect()
post :
(Downloadable.allInstances - Downloadable.allInstances@pre) ->
forall(dpa:Downloadable | 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.isDefined() then dpa.expiryDays = self.expiryDays
  else self.expiryDays = Download.allInstances->any(true).daysExpiryDelay
de
  and
  if self.maximumDownloadCount.isDefined() then
    dpa.maximumDownloadCount = self.maximumDownloadCount
  else self.maximumDownloadCount = Download.allInstances->any(true).maximumNumberOfDownloads
de
end)

context NewLanguage::effect()
post :
(Language.allInstances - Language.allInstances@pre) ->forall(l:Language |
  l.oclIsNew() and
  l.oclIsTypeOf(Language) and
  l.name = self.newName and
  l.code = self.newCode and
  l.defaultCurrency = self.defaultCurrency)

context NewManufacturer::effect()
post :
(Manufacturer.allInstances - Manufacturer.allInstances@pre) ->forall(m:Manufacturer |
  m.oclIsNew() and
  m.oclIsTypeOf(Manufacturer) and
  m.name = self.name and
  Language.allInstances ->forall(l |
    self.hasURL -> select(languageOfURL=l).url =
context NewNewsletter::effect()

post :
(Newsletter.allInstances - Newsletter.allInstances@pre) -> forAll(n:Newsletter |
  n.oclIsNew() and
  n.oclIsTypeOf(Newsletter) and
  n.title = self.title and
  n.content = self.content and
  n.status = #unlocked )

context NewOrderStatus::effect()

post :
(OrderStatus.allInstances - OrderStatus.allInstances@pre) -> forAll(os:OrderStatus |
  os.oclIsNew() and
  os.oclIsTypeOf(OrderStatus) and
  Language.allInstances->
  forAll(l|
    self.hasOrderStatusName
    =>$select(languageOfOrderStatus=l).orderStatusName.string=
    os.orderStatusInLanguage->
    select(language=l).name )

context NewProduct::effect()

post :
(Product.allInstances - Product.allInstances@pre) -> forAll(p:Product |
  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.modelM = self.modelM and
  p.imagePath = self.imagePath and
  p.weight = self.weight and
  p.category = self.category and
  p.manufacturer = self.manufacturer and
  p.taxClass = self.taxClass and
  Language.allInstances ->
  forAll (l|
    self.hasNewProductName =>$select(languageOfProduct=l).nameOfProduct.string =
    p.productInLanguage->select(language=l).name))

context NewProductAttribute::effect()

post :
(ProductAttribute.allInstances - ProductAttribute.allInstances@pre) -> forAll(pa:ProductAttribute |
  pa.oclIsNew() and
  pa.oclIsTypeOf(ProductAttribute) and
  pa.increment = self.increment and
  pa.sign = self.sign and
  pa.product = self.product and
  pa.attribute.option = self.option and
  pa.attribute.value = self.value)

context NewProductNotification::effect()

post :
(ProductNotification.allInstances - ProductNotification.allInstances@pre) -> forAll(n:ProductNotification |
  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 = #unlocked )
context NewProductNotificationSubscription::effect()
post : self.customer.explicitNotifications -> includes(self.newSubscribedProduct)

class NewProductOption::effect()
post :
{Option.allInstances - Option.allInstances@pre} -> forAll(po:Option |
pooclIsNew() and
po.oclIsTypeOf(Option) and
Language.allInstances ->
forall (| self.hasNewOptionName -> select(languageOfOption=|).nameOfOption =
op.hasOptionName -> select(optionLanguage=|).optionName)}

context NewProductOptionValue::effect()
post :
{Value.allInstances - Value.allInstances@pre} -> forAll(pov:Value |
povoclIsNew() and
pov.oclIsTypeOf(Value) and
Language.allInstances ->
forall (| self.hasNewValueName -> select(valueLanguage=|).valueName =
pov.hasValueName -> select(valueLanguage=|).valueName) and
pov.option = self.option)

context NewReview::effect()
post :
{Review.allInstances - Review.allInstances@pre} -> forAll(r:Review |
roclIsNew() 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)

context NewSession::effect()
post :
{Session.allInstances - Session.allInstances@pre} -> forAll(s:Session |
soclIsNew() and
s.oclIsTypeOf(Session) and
s.currentCurrency=self.currentCurrency and
s.currentLanguage=self.currentLanguage and
s.sessionID=Session.allInstances->size())

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

context NewTaxZone::effect()
post :
{TaxZone.allInstances - TaxZone.allInstances@pre} -> forAll(tz:TaxZone |
tzoclIsNew() and
tz.oclIsTypeOf(TaxZone) and
tz.name = self.name and
tz.description = self.description and
tz.zone = self.zone)

context NewTaxRate::effect()
post :
{TaxRate.allInstances - TaxRate.allInstances@pre} -> forAll(tr:TaxRate |
troclIsNew() 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)

context NewTaxClass::effect()
post :
(TaxClass.allInstances - TaxClass.allInstances@pre) -> forAll(tc:TaxClass | 
  tc.oclIsNew() and 
  tc.oclIsTypeOf(TaxClass) and 
  tc.name = self.name and 
  tc.description = self.description)

context NewZone::effect()
post :
(Zone.allInstances - Zone.allInstances@pre) -> forAll(z:Zone | 
  z.oclIsNew() and 
  z.oclIsTypeOf(Zone) and 
  z.name = self.name and 
  z.code = self.code and 
  z.country = self.country)

context OrderConfirmation::effect()
post theOrderIsCreated:
(Order.allInstances - Order.allInstances@pre) -> forAll(o:Order | 
  o.oclIsNew() and 
  o.oclIsTypeOf(Order) and 
  self.orderCreated=o 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 
  (OrderStatusChange.allInstances - OrderStatusChange.allInstances@pre) -> forAll(osc:OrderStatusChange | 
    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 -> exists 
        (olAtt|olAtt.orderLine = ol and 
         olAtt.attribute = iAtt)))))

post theShoppingCartIsRemoved:
ShoppingCart.allInstances->excludes(self.shoppingCart@pre)

post updateProductQuantities:
let productsBought:Set(Product) = 
  self.shoppingCart@pre.shoppingCartItem@pre->asSet()

in productsBought -> forAll (p|
  let quantityBought:Integer = 
    self.shoppingCart@pre.shoppingCartItem@pre->select
    (sc | sc.product = p).quantity -> sum()
  in
  p.quantityOrdered = p.quantityOrdered@pre + quantityBought and 
  Stock.allInstances->any(true).substractStock implies 
  p.quantityOnHand = p.quantityOnHand@pre - quantityBought)
context OwnerChange::effect()
    post : myStore().owner = self.newOwner

class PasswordChange
    context PasswordChange::effect()
    post : self.customer.password = self.newPassword

context PasswordMinimumChange::effect()
    post : MinimumValues.allInstances->any(true).password = self.newMinimum

context PercentageIncreaseForLargerPackagesShippingConfigurationChange::effect()
    post : ShippingAndPackaging.allInstances->any(true).percentageIncreaseForLargerPackages = self.newPercentage

context PostCodeMinimumChange::effect()
    post : MinimumValues.allInstances->any(true).postCode = self.newMinimum

context PostCodeShippingConfigurationChange::effect()
    post : ShippingAndPackaging.allInstances->any(true).postCode = self.newPostCode

context PrimaryCustomerAddressChange::effect()
    post : self.customer.primary = self.address

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

context ProductDownload::effect()
    post :
        let OrderDownloadFromProduct:OrderDownload = self.customer.order.orderLine.orderLineAttribute->select (ola | ola.oclIsTypeOf(OrderDownload) and ola.orderLine.product = self.product) -> asSequence() -> last() .oclAsType(OrderDownload)

    in
        let OldOrderDownloadCount:Integer = self.customer.order.orderLine.orderLineAttribute@pre -> select (ola | ola.oclIsTypeOf(OrderDownload) and ola.orderLine.product = self.product) -> asSequence() -> last() .oclAsType(OrderDownload).downloadCount

    in
        OrderDownloadFromProduct.downloadCount = OldOrderDownloadCount + 1

context ProductOptionAttributeChange::effect()
    post : productAttribute.attribute.option = self.option

context ProductValueAttributeChange::effect()
    post : productAttribute.attribute.value = self.value

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

context ReadProductInfo::effect()
    post : self.product.productInLanguage->select (pil | pil.language=self.language) ->any(true).viewed =
            self.product@pre.productInLanguage@pre->select (pil | pil.language=self.language) ->any(true).viewed + 1

context ReadReview::effect()
    post : self.review.timesRead = self.review@pre.timesRead + 1

context ReorderLevelStockConfigurationChange::effect()
    post : Stock.allInstances->any(true).stockReOrderLevel = self.newValue
context RestorePreviousShoppingCart::effect()
post : self.session.shoppingCart = self.customer.customerShoppingCart

context ReviewTextMinimumChange::effect()
post : MinimumValues.allInstances->any(true).reviewText = self.newMinimum

context SendExtraOrderEmailChange::effect()
post : myStore().sendExtraOrderEMail->includesAll(self.newSendExtraOrderEMail)

context SendNewsletter::effect()
post : true

context SetCancelledOrderStatus::effect()
post : self.myStore().cancelledStatus = self.orderStatus

context SetCurrentCurrency::effect()
post : self.session.currentCurrency = self.newCurrentCurrency

context SetCurrentLanguage::effect()
post :
  session.currentLanguage = self.newCurrentLanguage
post :
  Store.allInstances -> any(true).switchToDefaultLanguageCurrency and
  self.newCurrentLanguage.defaultCurrency -> notEmpty()
implies
(SetCurrentCurrency.allInstances - SetCurrentCurrency.allInstances@pre)
  -> forAll(ccc:SetCurrentCurrency | ccc.oclIsNew() and
  ccc.oclIsTypeOf(SetCurrentCurrency) and
  ccc.session = self.session and
  ccc.newCurrentCurrency = self.newCurrentLanguage.defaultCurrency)

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

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

context SetDefaultOrderStatus::effect()
post : self.myStore().defaultStatus = self.orderStatus

context ShowBanner::effect()
post :
  BannerHistory.allInstances -> one
  (bh | bh.banner = self.banner and
  bh.shown = bh@pre.shown + 1)

context StateCustomerDetailChange::effect()
post : CustomerDetails.allInstances->any(true).state = self.newValue

context StateMinimumChange::effect()
post : MinimumValues.allInstances->any(true).state = self.newMinimum

context StatusPaymentMethodChange::effect()
post : self.paymentMethod.status = self.newStatus

context StatusShippingMethodChange::effect()
post : self.shippingMethod.status = self.newStatus

context StoreAddressAndPhoneChange::effect()
post : myStore().storeAddressAndPhone = self.newStoreAddressAndPhone

context StreetAddressMinimumChange::effect()
post : MinimumValues.allInstances->any(true).streetAddress = self.newMinimum
context SubstractStockConfigurationChange::effect()
  post : Stock.allInstances->any(true).subtractStock = self.newValue

context SuburbCustomerDetailChange::effect()
  post : CustomerDetails.allInstances->any(true).suburb = self.newValue

context SwitchToDefaultLanguageCurrencyChange::effect()
  post : myStore().switchToDefaultLanguageCurrency = self.newSwitchToDefaultLanguageCurrency

context TaxDecimalPlacesChange::effect()
  post : myStore().taxDecimalPlaces = self.newTaxDecimalPlaces

context TelephoneMinimumChange::effect()
  post : MinimumValues.allInstances->any(true).telephoneNumber = self.newMinimum

context TypicalPackageTareWeightShippingConfigurationChange::effect()
  post : ShippingAndPackaging.allInstances->any(true).typicalPackageTareWeight = self.newValue

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

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

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

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

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

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

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

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

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

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

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

context UninstallTableRateShippingMethod::effect()
  post :
    TableRate.allInstances@pre->any(true).oclIsKindOf(OclAny)
context UninstallTwoCheckOutPaymentMethod::effect()
post: TwoCheckOut.allInstances@pre->any(true).oclIsKindOf(OclAny)

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

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

context UnlockNewsletter::effect()
post: self.newsletter.status = #unlocked

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

context UpdateOrderStatus::effect()
post: (OrderStatusChange.allInstances - OrderStatusChange.allInstances@pre)
-> forAll(osc:OrderStatusChange | osc.oclIsNew() and
osc.oclIsTypeOf(OrderStatusChange) and
osc.comments = self.comments and
osc.order = self.order and
osc.orderStatus = self.newOrderStatus)

context ZoneChange::effect()
post: myStore().zone = self.newZone

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.session.shoppingCart.shoppingCartItem->at(lc.index)
in (lc.remove or lc.quantity <> cartItem.quantity)
implies if lc.remove then
(RemoveProduct.allInstances
- RemoveProduct.allInstances@pre)
-> forAll(rp:RemoveProduct | rp.oclIsNew and
rp.oclIsTypeOf(RemoveProduct) and
rp.shoppingCartItem = cartItem)
else
(ChangeQuantity.allInstances -
ChangeQuantity.allInstances@pre)
-> forAll(cq:ChangeQuantity |
clc.oclIsNew() and
cq.oclIsTypeOf(ChangeQuantity) and
cq.shoppingCartItem = cartItem and
cq.quantity = lc.quantity)
endif )
APPENDIX B:
osCommerce methods

This appendix shows the methods of the domain events used in the CSTL program examples of section 9. The methods are also written using the CSTL language.

```cstl
method NameChange{
    self.myStore().name := self.newName;
}

method CountryChange{
    self.myStore().country := self.newCountry;
}

method InstallCreditCardPaymentMethod{
    cc:=new CreditCard;
    cc.status:=#enabled;
}

method InstallCashOnDeliveryPaymentMethod{
    cd:=new CashOnDelivery;
    cd.status:=#enabled;
}

method UninstallCreditCardPaymentMethod{
    delete CreditCard.allInstances->any(true);
}

method InstallPerItemShippingMethod{
    pi:=new PerItem;
    pi.status:=#enabled;
}

method InstallFlatRateShippingMethod{
    fr:=new FlatRate;
    fr.status:=#enabled;
}

method UninstallPerItemShippingMethod{
    delete PerItem.allInstances->any(true);
}

method NewLanguage{
    l:=new Language;
    l.name:=self.newName;
    l.code:=self.newCode;
    l.defaultCurrency:=self.defaultCurrency;
}

method EditLanguage{
    self.language.name:=self.newName;
    self.language.code:=self.newCode;
    self.language.defaultCurrency:=self.newDefaultCurrency;
}

method OrderConfirmation{
```
//The order is created
o:=new Order;
o.customer := self.shoppingCart.customer;
o.billing:=self.billing;
o.delivery=:self.delivery;
o.shippingMethod := self.shippingMethod;
o.paymentMethod := self.paymentMethod;
o.currency := self.currency;

//The initial status of an order is pending
OrderStatus os:=Store.allInstances->any(true).defaultStatus;
osc:=new OrderStatusChange(order:=o, orderStatus:=os);
osc.comments := self.comments;

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

//update product quantities
products:=o.orderLine.product->asSet();
Integer i:=0;
while products->size()>i do
  p:=products->asSequence()->at(i+1);
  subtract:= Stock.allInstances->any(true).substractStock;
  if subtract then
    var:=o.orderLine->select(product=p).quantity->sum();
  endif
  i:=i+1;
endwhile

//The shopping cart is removed
delete self.shoppingCart;
self.orderCreated:=o;

method PasswordMinimumChange{
  MinimumValues.allInstances->any(true).password := self.newMinimum;
}

method CreditCardNumberMinimumChange{
  MinimumValues.allInstances->any(true).creditCardNumber := self.newMinimum;
}

method AddressBookEntriesMaximumChange{
  MaximumValues.allInstances->any(true).addressBookEntries := self.newMaximum;
}
method GenderCustomerDetailChange{
    CustomerDetails.allInstances->any(true).gender := self.newValue;
}

method MaximumNumberDownloadConfigurationChange{
    Download.allInstances->any(true).maximumNumberOfDownloads := self.newMaximum;
}

method CheckLevelStockConfigurationChange {
    Stock.allInstances->any(true).checkStockLevel := self.newValue;
}

method TypicalPackageTareWeightShippingConfigurationChange{
    ShippingAndPackaging.allInstances->any(true).typicalPackageTareWeight :=
    self.newValue;
}

method MaximumPackageWeightShippingConfigurationChange{
    ShippingAndPackaging.allInstances->any(true).maximumPackageWeight :=
    self.newMaximum;
}

method StatusPaymentMethodChange{
    self.paymentMethod.status:=self.newStatus;
}

method EditCreditCardPaymentMethod{
    CreditCard.allInstances->any(true).splitCreditCardToMail :=
    self.newSplitCreditCardToMail;
    CreditCard.allInstances->any(true).status := self.status;
    CreditCard.allInstances->any(true).orderStatus := self.orderStatus;
    CreditCard.allInstances->any(true).taxZone := self.taxZone;
}

method EditPerItemShippingMethod{
    PerItem.allInstances->any(true).cost := self.newCost;
    PerItem.allInstances->any(true).handlingFee := self.handlingFee;
    PerItem.allInstances->any(true).taxZone := self.taxZone;
    PerItem.allInstances->any(true).taxClass := self.taxClass;
    PerItem.allInstances->any(true).status := self.status;
}

method SetDefaultLanguage{
    Store.allInstances->any(true).defaultLanguage := self.language;
}

method DeleteLanguage{
    delete self.language;
}

method NewCurrency{
    c:= new Currency;
    c.title:=self.title;
    c.code:=self.code;
    c.symbolLeft:=self.symbolLeft;
    c.symbolRight:=self.symbolRight;
    c.decimalPlaces:=self.decimalPlaces;
    c.value:=self.value;
    c.status:=#enabled;
}

method EditCurrency{
    self.currency.title:=self.newTitle;
    self.currency.code:=self.newCode;
    self.currency.symbolLeft:=self.newSymbolLeft;
    self.currency.symbolRight:=self.newSymbolRight;
    self.currency.decimalPlaces:=self.newDecimalPlaces;
}
self.currency.value:=self.newValue;
}

method DeleteCurrency{
    delete self.currency;
}

method SetDefaultCurrency{
    Store.allInstances->any(true).defaultCurrency:=self.currency;
}

method CurrencyStatusChange{
    self.currency.status := self.newStatus;
}

method NewCountry{
    c:=new Country;
    c.name:=self.name;
    c.isoCode2:=self.isoCode2;
    c.isoCode3:=self.isoCode3;
}

method EditCountry{
    self.country.name:=self.newName;
    self.country.isoCode2:=self.newIsoCode2;
    self.country.isoCode3:=self.newIsoCode3;
}

method DeleteCountry{
    Integer i:=0;
    while self.country.zone->size()>i do
        z:=self.country.zone->asSequence()->at(i+1);
        delete z;
    endwhile
    delete self.country;
}

method NewZone{
    z:=new Zone;
    z.name:=self.name;
    z.code:=self.code;
    z.country:=self.country;
}

method EditZone{
    self.zone.name:=self.newName;
    self.zone.code:=self.newCode;
}

method DeleteZone{
    delete self.zone;
}

method NewTaxZone{
    tz := new TaxZone;
    tz.name := self.name;
    tz.description := self.description;
    tz.zone := self.zone;
}

method EditTaxZone{
    self.taxZone.name := self.newName;
    self.taxZone.description := self.newDescription;
    self.taxZone.zone := self.newZones;
}
method DeleteTaxZone{
    delete self.taxZone;
}

method NewTaxClass{
    tc := new TaxClass;
    tc.name := self.name;
    tc.description := tc.description;
}

method EditTaxClass{
    self.taxClass.name := self.newName;
    self.taxClass.description := self.newDescription;
}

method DeleteTaxClass{
    delete self.taxClass;
}

method NewTaxRate{
    tc := self.taxClass;
    tz := self.taxZone;
    tr := new TaxRate(taxClass:=tc, taxZone:=tz);
    tr.rate := self.rate;
    tr.priority := self.priority;
    tr.description := self.description;
}

method EditTaxRate{
    tc := self.newTaxClass;
    tz := self.newTaxZone;
    tr := new TaxRate(taxClass:=tc, taxZone:=tz);
    tr.rate := self.newRate;
    tr.priority := self.newPriority;
    tr.description := self.newDescription;
    self.taxRate := tr;
}

method DeleteTaxRate{
    delete self.taxRate;
}

method NewProduct{
    p := new Product;
    p.status := self.status;
    p.available := self.available;
    p.netPrice := self.netPrice;
    p.modelM := self.modelM;
    p.imagePath := self.imagePath;
    p.weight := self.weight;
    p.category := self.category;
    p.manufacturer := self.manufacturer;
    p.taxClass := self.taxClass;
    Integer index := 0;
    while Language.allInstances->size()>index do
        l := Language.allInstances->asSequence()->at(index+1);
        hnpn := HasNewProductName.allInstances->select(languageOfProduct=l)
            ->select(productNameEvent=self)->any(true);
        pil := new ProductInLanguage(product:=p, language:=l);
        pil.name := hnpn.nameOfProduct.string;
        index := index+1;
    endwhile
}

method EditProduct{
self.product.status := self.status;
self.product.available := self.available;
self.product.netPrice := self.netPrice;
self.product.modelM := self.modelM;
self.product.imagePath := self.imagePath;
self.product.weight := self.weight;
self.product.category := self.category;
self.product.manufacturer := self.manufacturer;
self.product.taxClass := self.taxClass;

Integer i := 0;
while Language.allInstances->size() > i do
    l := Language.allInstances->asSequence() -> at (i + 1);
    hnpn := HasNewProductName.allInstances->select(languageOfProduct = l) -> select(productNameEvent = self) -> any(true);
    pil := self.product.productInLanguage -> any(language = l);
    pil.name := hnpn.nameOfProduct.string;
    i := i + 1;
endwhile

} method DeleteProduct{
    if self.product.orderLine->size() = 0
        then delete self.product;
    else
        new ProductStatusChange(product := self.product, newStatus := #outOfStock);
    endif
}

method ProductStatusChange{
    self.product.status := self.newStatus;
}

method NewProductOption{
    o := new Option;
    Integer i := 0;
    while Language.allInstances->size() > i do
        l := Language.allInstances->asSequence() -> at (i + 1);
        hnon := HasNewOptionName.allInstances->select(languageOfOption = l) -> select(productOptionNameEvent = self) -> any(true);
        oname := hnon.nameOfOption;
        pil := new HasOptionName(option := o, optionLanguage := l, optionName := oname);
        i := i + 1;
endwhile
}

method EditProductOption{
    o := self.option;
    Integer i := 0;
    while Language.allInstances->size() > i do
        l := Language.allInstances->asSequence() -> at (i + 1);
        hnon := HasNewOptionName.allInstances->select(languageOfOption = l) -> select(productOptionNameEvent = self) -> any(true);
        oname := hnon.nameOfOption;
        pil := new HasOptionName(option := o, optionLanguage := l, optionName := oname);
        hon := o.hasOptionName -> any(optionLanguage = l);
        delete hon;
        i := i + 1;
endwhile
}

method DeleteProductOption{
    Integer i := 0;
    valuesNotUsedSize := self.option.value->select(option->size() = 1)
    -> select(attribute.orderLineAttribute->isEmpty())->size() ;
while valuesNotUsedSize>i do
  v:=self.option.value->select(option->size()=1)
  ->select(attribute.orderLineAttribute->isEmpty())->asSequence()->at(i+1);
  delete v;
  i:=i+1;
endwhile
delete self.option;
}

method NewProductOptionValue{
  v:=new Value;
  Integer i:=0;
while Language.allInstances->size()>i do
  l:=Language.allInstances->asSequence()->at(i+1);
  hhnv:=HasNewValueName.allInstances->select(languageOfValue=l)
  ->select(productValueNameEvent=self)->any(true);
  vname:=hhnv.nameOfValue;
  new HasValueName(value:=v,valueLanguage:=l,valueName:=vname);
  i:=i+1;
endwhile
v.option:=self.option;
}

method EditProductOptionValue{
  v:=self.value;
  Integer i:=0;
while Language.allInstances->size()>i do
  l:=Language.allInstances->asSequence()->at(i+1);
  hnon:=HasNewValueName.allInstances->select(languageOfValue=l)
  ->select(productValueNameEvent=self)->any(true);
  oname:=hnon.nameOfValue;
  hon:=v.hasValueName->any(valueLanguage=l);
  pil:=new HasValueName(value:=v,valueLanguage:=l,valueName:=oname);
  delete hon;
  i:=i+1;
endwhile
v.option:=self.option;
}

method DeleteProductOptionValue{
  delete self.value;
}

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

method AttributeChange{
  o:=self.newOption;
  v:=self.newValue;
  pa:=self.productAttribute;
  attr:=Attribute.allInstances->select(value=v)->any(option=o);
  increment:=pa.increment;
  sign:=pa.sign;
  status:=pa.status;
  product:=pa.product;
  npa:=new ProductAttribute(product:=product, attribute:=attr);
  self.productAttribute:=npa;
  delete pa;
}

method IncrementAndSignAttributeChange{
pa:=self.productAttribute;
pa.increment:=self.newIncrement;
pa.sign:=self.newSign;
}

method DeleteProductAttribute
participantOrdersSize:=OrderLineAttribute.allInstances
->select(attribute=self.productAttribute.attribute)
->select(orderLine.product=self.productAttribute.product)->size();
if participantOrdersSize=0 then
    delete self.productAttribute;
else
    new ProductAttributeStatusChange(productAttribute:=self.productAttribute,
        newStatus:#disabled);
endif

method ProductAttributeStatusChange
    self.productAttribute.status:=#disabled;

method NewSpecial
    p:=self.product;
    s:=new Special;
    //We save the self.product information
    s.specialPrice:=self.specialPrice;
    s.expiryDate:=self.expiryDate;
    s.specialStatus:=self.status;
    s.status := p.status;
    s.available := p.available;
    s.netPrice:= p.netPrice;
    s.quantityOnHand := p.quantityOnHand;
    s.modelM:=p.modelM;
    s.imagePath:=p.imagePath;
    s.weight:=p.weight;
    s.category := p.category;
    s.manufacturer:=p.manufacturer;
    s.taxClass:=p.taxClass;
    Integer i:=0;
    while Language.allInstances->size()>i do
        l:=Language.allInstances->asSequence()->at(i+1);
        hnpn:=HasProductName.allInstances->select(languageOfProduct=l)
            ->select(product=p)->any(true);
        pil:=new ProductInLanguage(product:=s,language:=l);
        pil.name:=hnpn.nameOfProduct.string;
        i:=i+1;
    endwhile
    self.product:=s;
}

method EditSpecial
    self.special.specialPrice:=self.newSpecialPrice;
    self.special.expiryDate:=self.newExpiryDate;
    self.special.specialStatus:=self.newStatus;
}

method DeleteSpecial
    s:=self.special;
    p:=new Product;
    //We save the self.product information
    p.status := s.status;
    p.available := s.available;
    p.netPrice:= s.netPrice;
    p.quantityOnHand := s.quantityOnHand;
    p.modelM:=s.modelM;
    p.imagePath:=s.imagePath;
p.weight:=s.weight;
p.category := s.category;
p.manufacturer:=s.manufacturer;
p.taxClass:=s.taxClass;
i:=0;
while Language.allInstances->size()>i do
  l:=Language.allInstances->asSequence()->at(i+1);
  hnpn:=HasProductName.allInstances->select(languageOfProduct=l)
    ->select(product=p)->any(true);
  pil:=new ProductInLanguage(product:=p,language:=l);
  pil.name:=hnpn.nameOfProduct.string;
  i:=i+1;
endwhile
delete s;
}

method NewCategory
{
c:=new Category;
c.imagePath:=self.imagePath;
c.sortOrder:=self.sortOrder;
c.parent:=self.parent;
Integer i:=0;
while Language.allInstances->size()>i do
  l:=Language.allInstances->asSequence()->at(i+1);
  hncname:=HasNewName.allInstances->any(languageOfCategory=l).name;
  new HasCategoryName(category:=c,language:=l,categoryName:=hncname);
  i:=i+1;
endwhile
}

method EditCategory
{
c:=self.category;
c.imagePath:=self.imagePath;
c.sortOrder:=self.sortOrder;
c.parent:=self.newParent;
Integer i:=0;
while Language.allInstances->size()>i do
  l:=Language.allInstances->asSequence()->at(i+1);
  hnn:=HasNewName.allInstances->select(languageOfCategory=l)
    ->select(categoryNameEvent=self)->any(true);
  cname:=hnn.name;
  pil:=new HasCategoryName(category:=c,language:=l,categoryName:=cname);
  hcn:=c.hasCategoryName->any(language=l);
  delete hcn;
  i:=i+1;
endwhile
}

method MoveCategory
{
  self.category.parent:=self.newParent;
}

method DeleteCategory
{
topCategory:=self.category;
Integer i:=0;
while self.allChilds(topCategory)->size()>i do
  c:=self.allChilds(topCategory)->asSequence()->at(i+1);
  delete c;
  i:=i+1;
endwhile
delete self.category;
}

method MoveProduct
{
  newCat:=self.newCategory;
  oldCat:=self.oldCategory;
  categories:=self.product.category->union(Set{newCat})-Set{oldCat};
}
self.product.category:=categories;
}

method LinkProduct{
  newCat:=self.newCategory;
  categories:=self.product.category->union(Set{newCat});
  self.product.category:=categories;
}

method NewCustomer{
  c:=new Customer;
  c.gender:=self.primary.gender;
  c.firstName:=self.primary.firstName;
  c.lastName:=self.primary.lastName;
  c.dateOfBirth:=self.dateOfBirth;
  c.eMailAddress:=self.eMailAddress;
  c.phone:=self.phone;
  c.fax:=self.fax;
  c.newsletter:=self.newsletter;
  c.password:=self.password;
  c.numberOfLogons:=0;
  primaryAddress:=self.primary;
  c.address:=Set{primaryAddress};
  c.primary:=primaryAddress;
}

method PasswordChange{
  self.customer.password:=self.newPassword;
}

method NewManufacturer{
  m:=new Manufacturer;
  m.name := self.name;
  m.imagePath := self.imagePath;
  Integer i:=0;
  while Language.allInstances->size()>i do
    l:=Language.allInstances->asSequence()->at(i+1);
    hurl:=HasURL.allInstances->select(languageOfURL=l)->select(manufacturerURLEvent=self)->any(true);
    mil:=new ManufacturerInLanguage(manufacturer:=m,language:=l);
    mil.url:=hurl.url;
    i:=i+1;
  endwhile
}

method EditManufacturer{
  m:=self.manufacturer;
  m.name := self.name;
  m.imagePath := self.imagePath;
  Integer index:=0;
  while Language.allInstances->size()>index do
    l:=Language.allInstances->asSequence()->at(index+1);
    hurl:=HasURL.allInstances->select(languageOfURL=l)->select(manufacturerURLEvent=self)->any(true);
    mil:=m.manufacturerInLanguage->any(language=l);
    mil.url:=hurl.url;
    index:=index+1;
  endwhile
}

method DeleteManufacturer{
  m:=self.manufacturer;
  deleteProducts:=self.deleteProds;
  Integer ip:=0;
  if deleteProducts then
    while m.product->size()>ip do

\begin{verbatim}
p := m.product->asSequence()->at(ip+1);
p.status := #outOfStock;
ip := ip+1;
endwhile
endif
//Delete the manufacturer
delete m;
}

method NewBannerGroup{
bg := new BannerGroup;
bg.name := self.name;
}

method EditBannerGroup{
self.bannerGroup.name := self.newName;
}

method NewBanner{
b := new Banner;
b.title := self.title;
b.url := self.url;
b.imagePath := self.imagePath;
b.html := self.html;
b.expires := self.expires;
b.scheduled := self.scheduled;
b.status := #enabled;
b.bannerGroup := self.bannerGroup;
}

method EditBanner{
b := self.banner;
b.title := self.newTitle;
b.url := self.newUrl;
b.imagePath := self.newImagePath;
b.html := self.newHtml;
b.expires := self.newExpires;
b.scheduled := self.newScheduled;
b.status := self.newStatus;
b.bannerGroup := self.newBannerGroup;
}

method DeleteBanner{
delete self.banner;
}

method DeleteBannerGroup{
delete self.bannerGroup;
}

method NewNewsletter{
n := new Newsletter;
n.title := self.title;
n.content := self.content;
n.status := #unlocked;
}

method NewProductNotification{
n := new ProductNotification;
n.title := self.title;
n.content := self.content;
n.status := #unlocked;
n.global := self.global;
n.explicitNotifications := self.explicitNotifications;
}

method EditNewsletter{
\end{verbatim}
n:=self.newsletter;
  n.title:=self.newTitle;
  n.content:=self.newContent;
}

method DeleteNewsletter{
  delete self.newsletter;
}

method LockNewsletter{
  self.newsletter.status:=#locked;
}

method UnlockNewsletter{
  self.newsletter.status:=#unlocked;
}

method NewSession{
  s:=new Session;
  self.createdSession:=s;
  s.currentCurrency:=self.currentCurrency;
  s.currentLanguage:=self.currentLanguage;
  s.sessionID:=Session.allInstances->size();
}

method DeleteSession{
  delete self.session;
}

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

method AddProductToShoppingCart{
  //Shopping cart item is created
  sci:=new ShoppingCartItem;
  sci.quantity:=self.quantity;
  sci.product:=self.product;
  sci.attribute:=self.attribute;

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

method RestorePreviousShoppingCart{
  self.session.shoppingCart := self.customer.customerShoppingCart;
}

method LogOut{
  self.session.customer := oclEmpty(Set{Customer});
}

method NewReview{
  r := new Review;
  r.review := self.review;
  r.rating := self.rating;
  r.customer := self.customer;
  r.product := self.product;
  r.language := self.language;
  self.createdReview := r;
}

method EditReview{
  r := self.review;
  r.review := self.newReview;
  r.rating := self.newRating;
  r.customer := self.newCustomer;
  r.product := self.newProduct;
  r.language := self.newLanguage;
}

method DeleteReview{
  delete self.review;
}

method NewOrderStatus{
  os := new OrderStatus;
  osi := 0;
  while Language.allInstances->size() > osi do
    l := Language.allInstances->asSequence()->at(osi+1);
    osname := HasOrderStatusName.allInstances->select(languageOfOrderStatus = l)
    ->select(orderStatusNameEvent = self)->any(true).orderStatusName;
    osli := new OrderStatusInLanguage(orderStatus := os, language := l);
    osli.name := osname.string;
    osi := osi+1;
  endwhile
self.createdOrderStatus:=os;
}

method EditOrderStatus{
os:=self.orderStatus;
os.language:=oclEmpty(Set(Language));
i:=0;
while Language.allInstances->size()>i do
  l:=Language.allInstances->asSequence()->at(i+1);
osname:=HasOrderStatusName.allInstances->select(languageOfOrderStatus=l)->select(orderStatusNameEvent=self)->any(true).orderStatusName;
  osl:=new OrderStatusInLanguage(orderStatus:=os,language:=l);
  osl.name:=osname.string;
  i:=i+1;
endwhile
}

method DeleteOrderStatus{
os:=self.orderStatus;
if Order.allInstances.orderStatus->includes(os)
  then
    self.orderStatus.status:=#disabled;
  else
    os.language:=oclEmpty(Set(Language));
    delete os;
endif
}

method CancelOrder{
cancelledStatus:=Store.allInstances->any(true).cancelledStatus;
osc:=new OrderStatusChange(order:=self.order,orderStatus:=cancelledStatus);
}

method SetCancelledOrderStatus{
  self.myStore.cancelledStatus:=self.orderStatus;
}

method SetDefaultOrderStatus{
  self.myStore.defaultStatus:=self.orderStatus;
}

method SetCurrentCurrency{
s.set_currency:=self.newCurrentCurrency;
}

method SetCurrentLanguage{
s.set_language:=self.newCurrentLanguage;
switch:=Store.allInstances->any(true).switchToDefaultLanguageCurrency;
changeCurrency:= s.newCurrentLanguage.defaultCurrency->notEmpty();
if changeCurrency
  then
    if switch then
      currentCurrency:=s.newCurrentLanguage.defaultCurrency;
      new SetCurrentCurrency(session:=s, newCurrentCurrency:=currentCurrency) occurs;
  endif
endif
}

method UpdateOrderStatus{
s:=self.newOrderStatus;
osc:=new OrderStatusChange(order:=self.order, orderStatus:=s);
osc.comments:=self.comments;
}
method EditCustomerDetails{
    c:=self.customer;
    c.gender:=self.newGender;
    c.firstName:=self.newFirstName;
    c.lastName:=self.newLastName;
    c.dateOfBirth:=self.newDateOfBirth;
    c.eMailAddress:=self.newEMailAddress;
    c.phone:=self.newPhone;
    c.fax:=self.newFax;
    c.newsletter:=self.newNewsletter;
}

method EditCustomer{
    c:=self.customer;
    c.gender:=self.newGender;
    c.firstName:=self.newFirstName;
    c.lastName:=self.newLastName;
    c.dateOfBirth:=self.newDateOfBirth;
    c.eMailAddress:=self.newEMailAddress;
    c.phone:=self.newPhone;
    c.fax:=self.newFax;
    c.newsletter:=self.newNewsletter;
    c.password:=self.newPassword;
    c.globalNotifications:=self.newGlobalNotifications;
}

method NewCustomerAddress{
    ad:=new Address;
    ad.gender:=self.gender;
    ad.firstName:=self.firstName;
    ad.lastName:=self.lastName;
    ad.company:=self.company;
    ad.street:=self.street;
    ad.suburb:=self.suburb;
    ad.postCode:=self.postCode;
    ad.city:=self.city;
    ad.state:=self.state;
    ad.zone:=self.zone;
    ad.country:=self.country;
    adSet:=Set{ad};
    self.customer.address:=self.customer.address->union(adSet);
}

method EditCustomerAddress{
    changedAddress:=self.address;
    newAddress:=self.newAddress;
    oldAddresses:=self.customer.address;
    if oldAddresses->size()=1 then
        self.customer.address:=Set{newAddress};
        self.customer.address:=self.customer.address-Set{changedAddress};
    else
        self.customer.address:=oldAddresses->union(Set{newAddress});
        self.customer.address:=self.customer.address-Set{changedAddress};
    endif
}

method PrimaryCustomerAddressChange{
    self.customer.primary:=self.address;
}

method DeleteCustomerAddress{
    deletedAddress:=self.address;
    self.customer.address:=self.customer.address-Set{deletedAddress};
}

method NewProductNotificationSubscription{
previousSubscriptions:=self.customer.explicitNotifications;
newProduct:=self.newSubscribedProduct;
if self.customer.explicitNotifications->size()>0 then
    self.customer.explicitNotifications:=previousSubscriptions->union(Set{newProduct});
else
    self.customer.explicitNotifications:=self.newSubscribedProduct;
endif
}

method DeleteProductNotificationSubscription{
    deletedSubscription:=self.deletedSubscribedProduct;
    previousSubscriptions:=self.customer.explicitNotifications;
    self.customer.explicitNotifications:=previousSubscriptions-Set{deletedSubscription};
}

method EditGlobalNotifications{
    self.customer.globalNotifications:=self.newGlobalNotifications;
}

method DeleteCustomer{
    //Delete reviews of customer
    while self.customer.review->size()>0 do
        r:=self.customer.review->any(true);
        r.product:=oclEmpty(Set(Product));
        r.language:=oclEmpty(Set(Language));
        r.customer:=oclEmpty(Set(Customer));
        delete r;
    endwhile
    //Delete shopping cart if needed
    if self.customer.customerShoppingCart->size()>0 then
        delete self.customer.customerShoppingCart;
    endif
    //Delete customer or set it to disabled
    if self.customer.order->size()>0 then
        new CustomerStatusChange(customer:=self.customer, newStatus:=#disabled)
        occurs;
    else
        delete self.customer;
    endif
}

method CustomerStatusChange{
    self.customer.status:=self.newStatus;
}