Introduction

The assignment is about creating an IT system for a fictive bicycle shop, NBS (Noware Bicycle Shop). NBS specializes in selling customized handmade bicycles and have been in business for several years as an old-fashioned, physical shop.

The main problem they are having is that it is difficult to demonstrate the possibilities of customizing bicycles for the customers just through magazines and catalogs. Secondly, the sales clerks spend a great amount of their time on supporting customers to choose valid configurations for their bikes.

To alleviate the work of sales clerks and make it easier for the customers to customize bikes NBS would like to introduce an IT solution with touch screen terminals where customers can easily create their model of new bikes. The clerks will make use of a textual interface to rapidly create bike configurations.

Motivation

The group's motivation for creating this task is to use the modeling concepts learnt through the Model Driven Development (MDD) lectures and apply it on a realistic scenario. By the completion of the assignment the members will have a better idea of how MDD can be used to model a physical shop and apply its terms in an IT environment. The process will also give a general idea of how MDD can speed up the development process of the system.

The process

The report will describe the process of creating the system and most importantly the design decisions the group made to achieve an optimal mapping of the real world scenario to the IT system.

The point of departure is to create the meta model that represents a single customized bicycle. In order to do that the different levels of models and identify which level is the group working on, and which is meant to be for the users of the system.

To verify the validity of the created models we have to ensure they comply with the business and technical rules of NBS' customized bikes. For that purpose a set of OCL Constraints will be developed against which all the custom bicycles will be tested.

The next step is to create a graphical editor for the customers to use on the touch screens. //TODO: further discuss

As the sales clerks are more familiar with the buildup of bikes they are going to use a textual editor to define custom bikes to speed up their work.

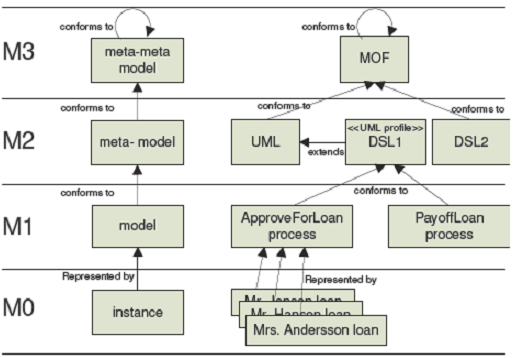
The final step is to provide the customer with an invoice which lists the bike configuration and customer details. This is done via Model to Text Transformations (Xpand) and is further discussed in the relating section.

Testing the system will be accomplished by creating three different bicycle models for three customers and using all the features of the system.

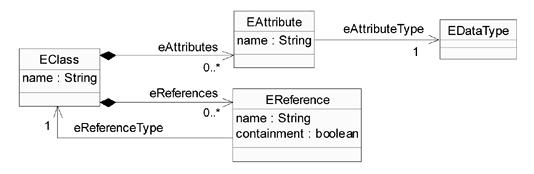
The tasks

Conceptual Model for a Custom Bicycle

Before creating the actual model of the customized bikes it is necessary to identify the model level we are working on.



MOF (Meta-Object Facility) provides a meta-meta model at the top layer (M3) to which all lower-level models conform.



The Ecore is meta-meta model is also based on MOF. It defines the elements used in the lower level (M2 meta-model). It provides the EClass which can have EAttributes. EAttributes have one EDataType. EReference can be used for referencing EClasses.

Having the M3 meta-meta model as a basis it is now possible to define the conceptual model for a bicycle at the M2 level.

NBS provided the following description of what they mean by a customized bike and the additional attributes it has:

NBS has specialized in bicycles of three base types: racers, city bikes and unicycles. The description of a single bicycle has a customer name, an address, a calculated price, and a wanted delivery date.

A bicycle is defined as follows:

• It has a frame – three types: racer, city bike or unicycle

• It has one or two wheels

• It can have an outer gear if it has two wheels and it does not have an inner gear

• It has pedals

• It has a saddle

• It has hand brakes if it has two wheels

• It has handlebars (Danish: “cykelstyr”) if it has two wheels

• The frame can be painted in any color

• All parts can be constructed by three different companies: “Super Parts”, “Home Parts” and

“Handmade Parts” – though only parts from the first two suppliers can be combined on a single bicycle.

The customer can choose between different saddles, pedals, etc. when defining a custom bike.

For instance, there could be four saddles to choose between; a racing saddle, a child saddle, a special unicycle saddle and a comfort saddle.

Each part has a price – that can be entered by the sales clerk, but not by the customer.

The group came up with the following conceptual model by deriving the previously mentioned requirements.



The main class of the model is the Part class. It is an abstract class and all the actual parts of the bicycle extend this. The abstract Node class was introduced to conform to the convention of having a Node Super Type presented by the exercises solutions during the course.

The Part has all the default attributes (price, manufacturer and id) that all the parts of the bicycle are supposed to have.

Enumeration types were used to represent the various types of parts NBS provides to their customers. These are defined in the FrameType, SaddleType, Color, PedalTypes, Manufacturers classes.

A note on the Color class; although the description states that the frame can be painted in any color enumerating the different colors was found to be the most suitable solution. In order to provide more flexibility to the users of the system a Color could have been represented by the three RGB values in a 0-255 range, but it was after discussing this option the group found it too circumstantial both to develop and the users to use as it would require an additional tool to provide an RGB value to a certain color.

A note on the InnerGear and OuterGear classes; as opposed to using enumerations in previous cases, this time subtypes were used for the reason that there are only two types of gears on bikes and it is not expected to change. This implies that the Gear class is an abstract class.

A note on the Bicyle, CustomerData and Parameter classes; the Bicycle class represents the customized bike itself which is built up from the different parts. It has a Price attribute which will be calculated by summing up the prices of individual parts. CustomerData captures the basic information about the customer ordering the bike and it was decided to be extracted from the attributes of Bicycle to have a better modularity and overview of the model. It is possible to extend the CustomerData and Bicycle classes via the Parameter class which has name-value pairs in case it is needed.

References from Parts, CustomerData and Parameter to Bicycle are of a composite relationship because all these information have to be connected to an instance of a Bicycle.

To sum up the Model part we need to mention the two additional meta levels. M1 level models are going to be created by the users of the system. Level M1 represents a model of a concrete bicycle ordered by a specific customer. Level M0 is a real world instance of the bicycle made for the customer.

OCL Constraints

NBS provided rules how to ensure that only valid bikes are modeled and ordered by customers. These were realized in form of OCL statements and implemented in the Ecore model as invariants too. Having them as invariants provides a much easier validation because the whole model can be validated at once against all constraints without having to type them into the console one-by-one.

The constraints are the following:

* It has one or two wheels.
* It has an outer gear if it has two wheels and it does not have an inner gear.
* If it has two wheels, then it must have hand brakes.
* As specified above: “All parts can be constructed by three different companies: “Super Parts”, “Home Parts” and “Handmade Parts” – though only parts from the first two suppliers can be combined on a single bicycle”.
* Define two additional constraints yourself.

The solution statements are listed in the appendix part.

Appendix

***OCL Constraints***

The Object Constraint Language (OCL) describes rules for a model. The statements below describe how custom bicycle has to look like.

1. It has one or two wheels

self.parts->select(oclIsTypeOf(Wheel))->size() > 0

and self.parts->select(oclIsTypeOf(Wheel))->size() < 3

The statement checks that the size of a sequence of elements with type Wheel is 1 or 2.

2. It has an outer gear if it has two wheels and does not have an inner gear

self.parts->select(oclIsTypeOf(Wheel))->size() = 2

implies self.parts->select(oclIsKindOf(Gear))->size() = 1

The statement checks that if a bicycle has two parts of type Wheel, it has to have also one part of any kind of gear.

3. If it has 2 wheels, it must have hand brakes

self.parts->select(oclIsTypeOf(Wheel))->size() = 2

implies self.parts->select(oclIsTypeOf(HandBrakes))->size() = 1

The statement checks that if a bicycle has two parts of type Wheel, it has to have also one part of type HandBrakes.

4. All parts must be from manufacturers "Super Parts" and "Home Parts" OR from manufacturer "Handmade Parts"

self.parts->select(manufacturer = Manufacturers::HandMadeParts)->size() = self.parts->size() or

self.parts->select(manufacturer = Manufacturers::HandMadeParts)->size() = 0

The statement checks that a bicycle has to have all parts from manufacturer HandMadeParts (amount of all parts = amount of parts from manufacturer HandMadeParts) or none of them (amount of parts from manufacturer HandMadeParts = 0).

Additional constraints:

5. A bike must have a frame

self.parts->select(oclIsTypeOf(Frame))->size() = 1

The statement checks that the size of a sequence of elements with type Frame is 1.

6. A bike must have a pair of paddles

self.parts->select(oclIsTypeOf(Pedals))->size() = 1

The statement checks that the size of a sequence of elements with type Pedals is 1.

7. A bike must have a saddle

self.parts->select(oclIsTypeOf(Saddle))->size() = 1

The statement checks that the size of a sequence of elements with type Saddle is 1.

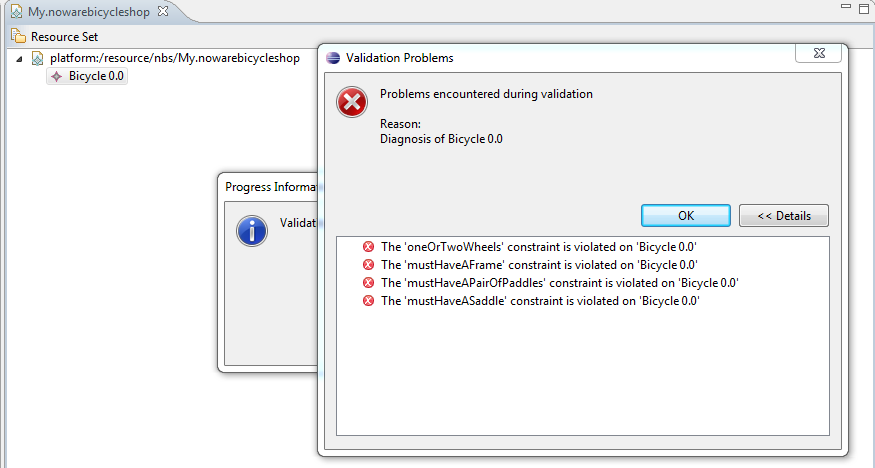
8. If it has 2 wheels, it must have handle bars

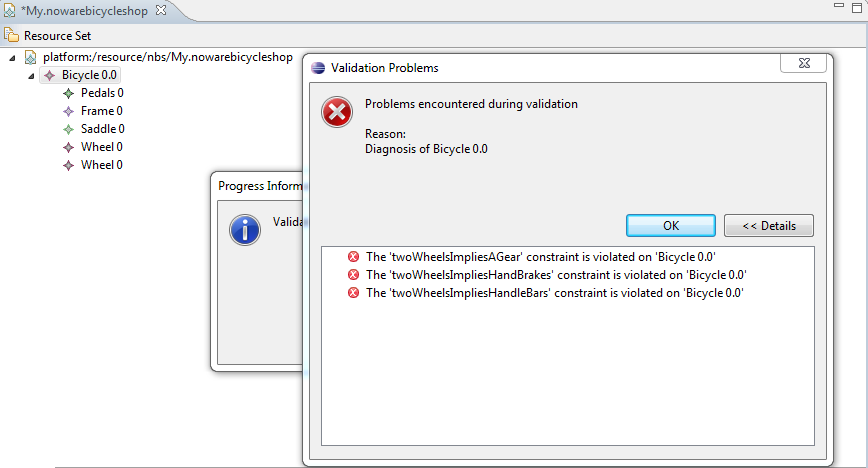
self.parts->select(oclIsTypeOf(Wheel))->size() = 2

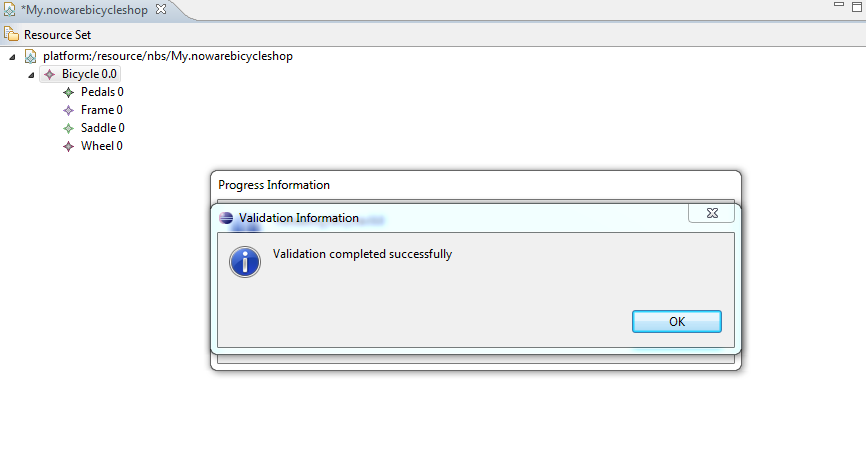
implies self.parts->select(oclIsTypeOf(HandleBars))->size() = 1

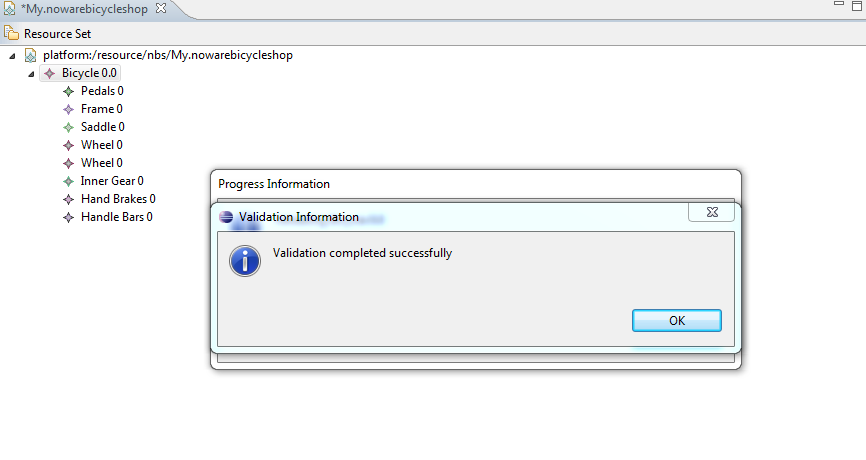
The statement checks that if a bicycle has two parts of type Wheel, it has to have also one part of type HandleBars.

All constraints are implemented in the ecore model as an invariants.

**

**

**

**

Pedals are from manufacturer "Super Parts" and a frame is from manufacturer "Handmade Parts", what is not allowed.

