FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Ensuring DEMO Model Consistency for the OpenPonk Platform
Student:	Bc. Jakub Nováček
Supervisor:	Ing. Robert Pergl, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2019/20

Instructions

1. Acquaint yourself with the DEMO methodology, the Smalltalk programming language, the OpenPonk conceptual modelling platform (OP, http://openponk.github.io/), its architecture and current implementation of the DEMO diagramming.
2. Devise a UML profile for DEMO diagrams and use it to implement DEMO diagrams in OP.
3. Propose and implement a set of diagram consistency checks and modelling aids and implement them in OP.
4. Demonstrate and comment your achievements and formulate guidelines for analysts.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague September 25, 2018

Insert here your thesis' task.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Ensuring DEMO Model Consistency for the OpenPonk Platform

Bc. Jakub Nováček

Department of Software Engineering
Supervisor: Ing. Robert Pergl, Ph.D.

February 15, 2019

Acknowledgements

I would like to thank to my supervisor Ing. Robert Pergl, Ph.D. and to Ing. Peter Uhnák for introduction into OpenPonk and related answers.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on February 15, 2019 ....

Czech Technical University in Prague
Faculty of Information Technology
© 2019 Jakub Nováček. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Nováček, Jakub. *Ensuring DEMO Model Consistency for the OpenPonk Platform.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato diplomová práce je zaměřena na sestavování UML profilů realizujících DEMO diagramy. Jsou odvozeny metamodely diagramů a následně celkový metamodel DEMO a odpovídající UML profily.

Klíčová slova DEMO, UML, UML Profile, UML Profiles, Pharo, OpenPonk

Abstract

This thesis is focused on composing UML profiles realizing DEMO diagrams. Metamodels of diagrams are derived and DEMO metamodel and corresponding UML profiles subsequently.

Keywords DEMO, UML, UML Profile, UML Profiles, Pharo, OpenPonk

Contents

List of Figures

Introduction

There is not many satisfying modelling tools for DEMO methodology. With growing possibilities of OpenPonk platform there is a possibility to implement such tool. A newly implemented basis for UML profiles offers even an approach to simplify logic of DEMO diagramming regarding consistency between diagrams. The main goal of this thesis is to design and implement UML profiles for DEMO diagramming in OpenPonk platform in such a way that every diagram or table are a view on only one common model.

Chapter 1

Platform and Theory

1.1 Pharo

Pharo is an object-oriented programming language, highly influenced by Smalltalk. (Bergel et al., 2013)

1.2 OpenPonk

OpenPonk is a modelling platform implemented in the dynamic environment Pharo aimed at supporting activities surrounding software and business engineering such as modelling, executing, simulation, source code generation, etc. (Uhnak, 2018)

OpenPonk provides tools for modelling in several modelling notations: BORM, UML, DEMO. Having partially implemented UML profiles the focus moves on implementing DEMO diagramming using UML profiles instead of improvements of the current DEMO tool.

OpenPonk platform benefits from MVC architecture. View part uses Roassal 2 as a visualisation engine written in the Pharo with Trachel for drawing graphical elements. (Agilevisualization.com, 2019)

1.3 UML Profile

UML specification defines Profiles that serves an adaptation of UML for different purposes. Metaclass extension as it is allowed in a Profile enables tailor the UML metamodel for different platforms or domains. This mechanism provides adaptations an existing metamodel grouped into a Profile and it is possible to add new constraints that are specific to the Profile. It is worth to note that it is not possible to remove any of those constraints that exist as defined for UML, only add new ones.

Before we look at possibilities of the UML Profile, it is important to understand the different conceptual levels where UML Profile corresponds to one level and DEMO diagrams to another one. As it is defined in OMG metamodeling (Object Management Group, 2017) there are 4 levels. Contrary to OMG, let's take them in reverse order:

- Level M0, contains run-time instances of the model elements defined in a model, Instance Models
- Level M1, defines a language that describe semantic domains, User Models
- Level M2, defines a language for specifying models, e.g. UML
- Level M3, defines a language for specifying a metamodel, e.g. MOF

From this point of view, concrete implemented instances are at the level M0 and DEMO diagrams describing some organization are at the level M1. If our intention is to define DEMO language (no matter whether as an extension of UML or not) then it is one level above the level of DEMO diagrams – the level M2. There on the level of UML we can create an extension as a UML Profile which is intended to specify DEMO diagram. The highest level in this model hierarchy – M3 – is the level of MOF, which stands for Meta Object Facility, a language to define modelling languages, such as UML (see UML as an instance of MOF).

The elements of the M1 level are models. As an element of the level M2 is the model of the model, it is called metamodel and analogously, an element of the level M3 is called meta-metamodel.

UML extension using a Profile enables to give a terminology that is adapted to a particular platform or domain, give a syntax for constructs that do not have a notation, give a different notation for already existing elements, add additional semantics to UML or specific metaclasses, add types that do not exist in UML, add constraints that restrict the way UML's constructs are used, add information that can be used when transforming a model to another model or code. (Object Management Group, 2017)

These Profile features can be used in our context of DEMO metamodelling to use terms of the DEMO methodology, for example we can have an element Transaction directly defined in a modelling language to represent a transaction, to have appropriate graphical representations, for example a diamond in a circle to represent a transaction, to have a constraints restricting UML into a frame that we need to be in accordance with DEMO specification, for example constraint for transaction that at least one association to an initiater and exactly one association to an executor is mandatory, etc.

1.3.1 Profile Extension Mechanism

The primary extension construct is the stereotype. Metaclass is a class which may be extended through one or more stereotypes. Stereotype is a profile class that defines such metaclass extension as part of a profile. It is possible to define additional meta-information for a stereotype, tag definitions. Such metaproperties of a stereotype can have a default value.

Definition of stereotypes is what we need in order to establish the elements of the DEMO methodology, such as transaction, actor, product. For example, representation of Product is possible using Metaclass Class extended as a stereotype Product.

As the notation for icon presentation is specified, it is possible to have the appropriate graphical representation of the DEMO methodology elements by attached Image to a stereotype (using composition arrow). When such stereotype has been applied on some model element, the element can be graphically represented by the Image or the Image can be used in addition to the normal notation of the element.

Profiles are also endowed an ability to restrict availability of elements. It will be seen further below how such ability can be used to make certain elements of DEMO metamodel visible and other ones hidden to extract appropriate representations into given DEMO diagram and leave what is not relevant for the diagram. Restricting availability is reached by profile's filtering rules. These rules distinguish whether an element shall be available or hidden after profile application. The rules may be specified in metaclassReference ElementImports and metamodelReference PackageImports. In order to make the rules activated, the profile must be applied in Strict mode (the attribute isStrict on the ProfileApplication must be set on true). The model element (one of metaclasses and their instances) is available when at least one of the following conditions is fulfilled. Otherwise element is hidden.

- Is referenced by an explicit metaclassReference
- Is a member of a package that is referenced by an explicit metamodelReference and metaclassReference is absent
- Is extended by a stereotype which is a member of the applied profile

(Object Management Group, 2017)

1.3.2 Other Profile Possibilities

It has been specified (Object Management Group, 2017) that it is possible to apply several profiles to the same package. Further it is possible to extend a profile by another profile, which allows a specialization of a stereotype as well as referencing. These both are key abilities to form the solution that will be seen in the next chapter.

Chapter 2

Building UML Profiles

In order to build profiles for the DEMO diagrams, it is neccessary to have a metamodel the profiles will be derived from.

2.1 DEMO Metamodel

As clearly written in the UML Specification (Object Management Group, 2017), applying a profile to a model does not change that model in any way; it merely defines a view of the underlying model. This is basically the main idea of DEMO diagramming realization as targeted in this thesis, the main idea of ensuring diagrams consistency. Because of that the realization of DEMO diagrams as UML profiles is what enables the consistency throughout all the diagrams. DEMO methodology is built to represent a part of the real world so there is consistency of the DEMO diagrams themself, of course. Both means that it is possible to have one complete metamodel where its concrete model holds all the information represented by all the diagrams and the diagrams are realized as a profile.

2.1.1 DEMO Metamodel Construction

Taking the basic diagram Organisation Construction Diagram (OCD) and unrolling to the other diagrams the metamodel consists of the following elements, relationships, constraints:

- OCD defines an actor role and a transaction kind with a relationship between them. As a transaction has one or more initiators and one executor there are two such relationships.
- Transaction Product Table adds a product kind to the transaction kind. (Means adding another element and a relationship between them.) Each transaction has just one product – one-to-one relationship.

- Bank Contents Table adds a fact kind to the transaction kind (one(transaction)-to-many, optional).
- Actor Organisation Matrix adds an organisation role to the actor role (many-to-many, mandatory).
- Object Fact Diagram (OFD) adds a fact kind to the product kind. A product may be in relationship with a fact, a fact may be in relationship with another fact.
- Process Structure Diagram (PSD) enriches the transaction kind with a step kind. Such composition pointing to itself represents transition from one step in a transaction to another one.
- Action Rule Specification adds an action rule to the composition object of transaction kind + step kind mentioned in the previous point. The action rule contains complex information and therefore is a candidate for break up into more objects during implementation of a specific tool for operating with action rules (not part of this thesis) – the action rules are composed typically as a text.

Derived Fact Specification (a component of Fact Model next to OFD) and Work Instruction Specifications (a component of Action Model next to ARS) are not part of this thesis.

Transaction Process Diagram is fully derivable from PSD. Therefore information from this diagram is included implicitly.

2.2 Profiles layout

The main profile is DEMO profile for all the diagrams. All the other profiles – specific for concrete diagram, table and the like – are childs of the DEMO profile. It is possible by using the extensioning a profile by another profile allowed in UML profile specification as described in the previous chapter.

2.2.1 DEMO Profile

The main DEMO profile contains all the elements, relationships, constraints derived from the metamodel constructed above. That implicates all the information of model needed for arbitrary diagram is handled in this basic profile. Exceptions are stated under the metamodel construction points.

2.2.2 DEMO OCD Profile

The basic profile for Organisation Construction Diagram (OCD) is composed of stereotypes actor and transaction with relationship between them. The profile is a child of the DEMO profile.

2.2.3 DEMO TPT Profile

The Transaction Product Table (TPT) profile is composed of stereotypes transaction and product with relationship between them. The profile is a child of the DEMO profile. Implementation note: Because the TPT is a table and a model of the profile results in a diagram in the editor, at the front-end side the future layer between the profile and the editor will ensure that the data are arranged into a table. That layer – as a front-end part – is not part of this thesis.

2.2.4 DEMO PSD Profile

Process Structure Diagram (PSD) is a specific one from the UML profile point of view and it is to discussion whether it is to realization by UML profile like other diagrams or it would be beneficial to build specific tool to operate with that kind of diagram.

The basic part of the diagram is composed from transactions and actors which is information already provided from OCD (supposing using this tool in the expected order, i.e. OCD first). The second part is specific information of which collection is basically purpose of this diagram: Transitions between steps of transactions. Such transition can be defined as a pair of a couple transaction with step. (For example, T1/pm, T2/rq represents a promise of the transaction T1 is a cue for a request of the transaction T2.)

The differences from UML are as follows:

- In the basic part there is the actor role that is not presented as an usual element but as swimline.
- Instead of two relationships between the actor role and the transaction kind, the relation is represented as a line between transactions that goes through an actor – swimline.
- There is certain meaning in order of the lines between transactions and in position of the ends of the lines connected to the transaction elements.

The corresponding discussion points:

- Although the UML profile specification defines how to assign an icon to a stereotype it could be a significant claim that the swimline is a graphical representation of the actor role element.
- One option is to look at the relation as it is seen – between transaction kind elements. The second option is to look at the line like they are two lines – two more ends at the actor role element – swimline.
- As UML profile defined it is possible to establish new restrictions to a profile which could be interpreted that the order and positions of the line ends are restrictions defined for the given profile.

In case of a specific tool for PSD it is not the only one different from the standart profile for other diagrams, there are also tables (Transaction Product Table, Bank Contents Table) as differently viewed profiles and Action Rule Specification which is basically presented by structured text (but with references to elements presented in other profiles (like objects, step kind, transaction kind with step kind).

Another thing to consider is future development of the diagram specification in methodology that may bring another requirements not possible to realize using the specified UML profile.

2.2.5 DEMO BCT Profile

The profile for Bank Contents Table (BCT) is composed of stereotypes transaction and fact with relationship between them. The profile is a child of the DEMO profile.

Implementation note: Because the BCT is a table and a model of the profile results in a diagram in the editor, at the front-end side the future layer between the profile and the editor will ensure that the data are arranged into a table. That layer – as a front-end part – is not part of this thesis.

2.2.6 DEMO AOM Profile

The profile for Actor Organisation Matrix (AOM) is composed of stereotypes actor role and organisation role with relationship between them. The profile is a child of the DEMO profile.

Implementation note: Because the AOM is a table and a model of the profile results in a diagram in the editor, at the front-end side the future layer between the profile and the editor will ensure that the data are arranged into a table. That layer – as a front-end part – is not part of this thesis.

2.2.7 DEMO OFD Profile

The Object Fact Diagram (OFD) profile is composed of stereotypes product and fact with relationships. The profile is a child of the DEMO profile. DEMO ARS Profile

No description for the Action Rule Specification (ARS) profile as the ARS will have own tool and possibly realized without using UML profile.

Where the description of the relationship restrictions is not sufficient, see chapter DEMO Metamodel Construction, the appropriate point corresponding to the given profile by the name of the diagram/table.

Chapter 3

OpenPonk Platform Adjustments

3.1 OpenPonk Current State

During model creation the data of a model are stored in a class OPUMLModel as an array packagedElements. No matter which element of a profile it is, every element is stored as a class OPUMLClass. Stereotype that is realized by that is distinguished by tag and can be set by a method applyStereotype of a class OPUMLMetaElement which is superclass of the class OPUMLElement, superclass of the OPUMLClass.

Project menu containing items to load and save a project is defined in a class OPWorkbenchToolbar. Options regarding diagrams (open, save, etc.) will most probably be realized by a class OPDslEditor (the editor menu does not work for now). Once a user has selected a project to load in the open dialog window, project loads using OPProjectController by classes OPZipPersistence and OPPersistanceProjectReader. The project controller ensures opening the loaded project by sending the project to OPWorkbench.

3.2 Profile Creation

The profile is realised by a class OPUMLProfile. Attributes typically set to a profile:

- Name, UUID, URI, implementationPackage, implementationPrefix
- packagedElements
- (ownedStereotype, profile for owned stereotypes)

The name represents the name of the profile (will be set to ‚DEMO', ‚DEMO OCD', ...) which will be used in the profile switch menu to display and is also used programmatically to identify a profile. The implementationPrefix is determining names of stereotype classes for such profile in the phase of generation and is further used to approach them. The attribute packagedElements is to hold all the stereotypes corresponding to the profile (in a form of pairs where one element of the pair is OPUMLStereotype and the second one is OPUMLExtension). OwnedStereotype is to set the profile for every stereotype of the profile.

When a profile is generated the following classes and methods are created:

- Classes for every defined stereotype named by the chosen prefix (implementationPrefix set to the profile) and the chosen name (the name of the stereotype defined during creating an instance by instance creation method of OPUMLCustomProfile)
- with access methods for base class and initialization methods initializeDirectGeneralizations, initializeSharedGeneralizations.
- Access methods for every defined stereotype as an extension for the corresponding base class where the stereotype is pointing to (e.g. OPUMLClass)

Before generating any profile it is assumed that there is an element class which will be used as a superclass for the sterotype classes. This class has to have a name composed from the prefix defined for the profile and „Element“ and methods umlClassName and umlMetaClass.

3.3 OpenPonk platform extending and fixing

In order to implement the solution in OpenPonk platform it is neccessary to have functioning the parts of OpenPonk that matters and extend the platform by features that are not implemented yet.

The following fixing and extending of the platform is needed:

- Profile generation, there is a bug in the profile generation causing that the representation classes of the created profile are in some of the cases uncomplete; the diagram editor is not working with the profile correctly; that is to be fixed
- Strict mode, the strict mode (see Profile Extension Mechanism) is not implemented; there are basic attributes like isStrict flag for ProfileApplication class so it is possible to apply a profile in the strict mode but the diagram editor behaviour is the same as when the profile is applied without the strict mode: All the elements are available no matter whether they should be visible or not in accordance with the profile; that is to be implemented; Also to fully realize the intention of use of profiles it

is necessary to allow the switch of profiles – currently once diagram creation has begun, the profile switching is disabled and the only one that is chosen is available in the profile switching menu

- Constraints, there is no implementation of constraints for UML profiles; that is to be implemented

3.3.1 Profile Generation

Using profile generation on a profile where more stereotypes point to one metaclass it can be revealed that the profile generation has been failing in that case. When the method OPUMLProfileGenerator»generateExtensionMethods generate of profile factory for the profile generation is called with a profile comprised one stereotype pointing on one metaclass only, the mechanism sucessfully generates the profile that is needed. When the method generate is called with a profile comprising more stereotypes that point to one metaclass, the generation is not complete and after opening UML Class Diagram Editor in OP, setting the profile and trying to create the appropriate element an error message appears as the corresponding method was not found. Only one element (one that was defined in the profile as the last one) can be created successfully.

Comparing the state before and after the generation demonstrates what code was generated and considering the elements contained in the profile to generate it comes out that certain parts of the code that should be generated are missing. It can be seen in the Changes Browser at the end of generation as well.

As described above the profile generation consists of the classes representing defined stereotypes and access methods – extensions for the base class related to the stereotype. By comparison of the state before and after the generation or checking the changes in the Changes Browser it can be revealed that the missing parts are the extension methods for the base class (OPUMLClass in this case). Responsibility to generate those extension methods is given to the class OPUMLProfileGenerator, method generateExtensionMethods. Looking into that method it can be seen that the method sets an extensionGetter and an extensionSetter. The getter and the setter are added for a newly created class that is placed into a collection. As it is proceeded in a cycle for every element and the collection is managed as a dictionary (a map in Pharo) with the created classes names (that are derived from the metaclass name) as a key of the dictionary the error occurres when more elements are of the same metaclass because the name of the class is derived from the name of the metaclass so the next class with the same metaclass rewrites the previous one in the collection. This leads to the missing extension methods – getters, setters mentioned above and so non-working element in the diagramming.

The natural resolution is to condition the element inserting into the dictionary. The getters and the setters are always set but the class is created and inserted into the collection on the first time and then it is not created but taken the existing element from the collection. That will result in a metaclass with more extension methods – methods for every asociated class and it is possible to check that it is a desirable way by looking into a metaclass of any profile where more stereotypes are pointing to one metaclass, for instance OPUMLClass contains extension methods extensionClock for an example profile Clock and extensionMoment for OntoUML Profile.

In order to test such implementation it is verified in UML Class Diagram Editor that the classes are possible to create and all the extension methods are available in OPUMLClass next to the other extension methods of other profiles.

3.3.2 Strict Mode

Let's create an abstraction to simplify into the clear essence of the issue. Let's see a model as some set of elements where an element has defined a name and is of certain type. A profile application in strict mode results in viewing subset of the elements established in the model. A profile application is certain action with a profile and a model and so more profiles enables different views on a model – one profile application in strict mode enables to view certain subset of the elements of the model and another one enables to view a different subset.

For example, let's have a model M = T:a, U:b, V:c, V:d (where the big letter of each pair represents type – stereotype of the element and the small one represents the name of the element) and profiles P1(T,U), P2(U,V). Then profile application with strict mode will result in:

P1: a, b

P2: b, c, d

So we will see the elements a and b through the profile P1 but b, c, d through the profile P2.

Behind that, there is still the same model. So when b is changed it is changed in the model not in the profile which means no matter what profile we use the element will be seen changed there:

P1: a, b

b $\rightarrow$x (action: b is renamed to x)

P1: a, x

P2: x, c, d

Let's be aware what does it mean in the context of diagramming. When a model is opened under certain profile it is possible to see certain part of the model that relates to that profile. Doing changes in diagram editor has impact on the model and the change is seen in the editor. When switching the profile to another one still the change is there (if the given element is a component of the chosen profile so it is to show).

For example, when a DEMO model is opened in OCD profile it is possible to see actors, transactions and after renaming a transaction the transaction is renamed in the DEMO model so we see the changed transaction through the OCD profile. After opening OFD profile we will see the new name of the transaction as the view goes still through the same model.

3.3.2.1 Implementation in OpenPonk

In order to implement Strict Mode and related functionality the following extensions are needed:

- Show only the elements allowed in the strict mode instead of all the elements of the model when the sctrict mode is enabled
- Enable switch between DEMO profiles when any of the DEMO models is opened

Looking into important classes of OpenPonk (Controllers, OPWorkbench) it is possible to find key method used during loading model into the editor for diagramming allShowableElementsInModel of a class OPUMLPackageDiagramController which applies certain rules to the elements of a given model; taking a model returns a collection of the elements. Extension of that method by filtering the elements whether or not they are to show in accordance with the profile is one of the implementing points of the strict mode.

In the same controller OPUMLPackageDiagramController, there is a method descriptionAppliedProfile responsible for a component of the diagram editor – option switcher – control of profile applications. There is also implemented behaviour that enables to choose a profile on the beginning but freeze the choosen profile when any element is already inserted into a diagram in editor. Adjustment of that logic enables desirable switch between DEMO profiles when there are already loaded/inserted elements in diagram.

Another point is to add refresh of the diagram editor when a profile is switched as the profile switcher had been operating only on the beginning before composing a new diagram or with opening a model and profile switching was disabled since that so there was no need to refresh. That is implemented in the same method as the above point with profile switching component behaviour.

With the above described implementation it is possible to apply DEMO profiles in the strict mode. Realization of such profile application means that when applying a profile the ProfileApplication class has set flag isStrict to true. As it is not desirable to not have applied a DEMO profile in strict mode, this is bound to the option of every DEMO profile in profile switching menu so it is implemented directly in the method descriptionAppliedProfile as well as the last points of the strict mode implementation.

In order to verify how the implemented strict mode is fulfilling UML Profile specification regarding effects of unavailability of elements a comparison is following (Object Management Group, 2017):

- UML Specification: It is not possible to create new instances of that metaclass (or its subclasses).

  Implemented: When a profile is chosen in menu palette (menu with possible elements to create) is rebuild so there are only buttons for new instances of metaclasses that are part of the chosen profile.

- UML Specification: Existing instances of that metaclass (or its subclasses) can no longer be seen in diagrams or selected in lists, including browser panes.

  Implemented: Existing instances of metaclasses that are not part of the chosen profile in profile switcher are no longer in editor. But contrary to the specification it is possible to choose an element in a list on the left side of the editor in Model Tree.

- UML Specification: Relationships with existing instances of that metaclass (or its subclasses) can no longer be seen in diagrams or selected in lists, including browser panes.

  Implemented: With the elements to hide also the corresponding relationships are hidden.

The comparison suggests that there are two things to additional completion to fully fulfill UML Profile specification. The first one is adjustment for the list Model Tree so an unavailable element is not possible to choose. This Model Tree functionality is minor and most of the users will probably use different ways how to realize what they need. The implementation of that adjustment can be done in a class OPModelNavigator. The second thing to complete is to verify the behaviour of the editor when a metaclass has subclass(es) that is/are part of the used profile. To ensure that unavailability is in accordance with the specification also for subclasses not that unavailability is given by a membership in the used profile it is possible to add a check into creating a profile in OpenPonk UML profile generator, class OPUMLProfileGenerator so a created profile will be without cases where a metaclass is contained in a profile and its subclass is not.

Referencing profile extension mechanism as described in the previous chapter there are three conditions when an element is available. The implemented strict mode is based on the third one – an element is extended by a stereotype which is a member of the applied profile. This condition is the only one that is used for creating all the DEMO profiles. The first condition would have no effect on any of the currently established profiles as there is no metaclassReference used. The second condition is with the same effect because the only profile that uses metamodelReferences has profile application attribute isStrict set on false.

3.3.3 Constraints

Because the DEMO metamodel can not consist of only a list of stereotypes like actor, transaction, product, there is also a need for certain rules how to use the stereotypes, what kinds of elements may be connected between each other. For instance, an actor may be connected with a transaction and a transaction may be connected with a product but an actor may not be connected with a product or with itself.

Looking at such restrictions from the UML Profile point of view they are constraints (Fuentes and Vallecillo, 2004) (Object Management Group, 2017).

The important kind of constraint is, like already mentioned above, what elements (classes with applied stereotypes) may be connected with each other: An actor may be connected to a transaction, an actor may not be connected to a product. Another constraint is multiplicity: An actor may be connected to more transactions, a product can not be connected to more transactions.

3.3.3.1 Implementation

Implementation of constraints in OpenPonk relates to the implementation of profile. Constraints for the DEMO diagrams should be defined in the DEMO profiles. There are no constraints defined for currently generated profiles – OntoUML Profile and examples ClClock Profile, IPIssues Profile – and the class OPUMLProfile is not adjusted to creating such constraints. Therefore constraints functionality is added into the OPUMLProfile so it is possible to create a profile with constraints. The constraints are limited to the first above mentioned point – possible relationships between elements, multiplicity is not part of this thesis. The definition of the constraints for a new profile to create will be emplaced into definition of a profile (typically method createProfile). That is done for DEMO profiles as will be seen below – elements that are not allowed to be connected as DEMO methodology has specified are not allowed to be connected in an OpenPonk diagram.

Another implementation point to realize the constraints is the logic of checking elements in the class editor during diagramming when one element is selected and another one is to be chosen. That is done in a method can-BeTargetFor: of a controller class OPUMLClassController which is to check whether a class to be chosen by user as the target class in the relationship can be chosen with regard to the one that is as the source class – the class that was chosen by user as the first one. Because the checking logic needs access to the profile where the constraints are available and the check is directed from the class controller OPUMLClassController (the class controller has information about the class, the stereotypes applied but not about the profile itself), the access to the chosen profile is temporarily resolved by a new global variable #LastUMLProfile. That variable is set when a profile is chosen in the profile switcher and is accessed from the class controller to check the constraints. Such resolution is insufficient because of two reasons. Not only the global variable is against usual conventions but also this way, when there are more class editors opened, the last profile of one opened editor is not distinguished from the last profile of another one so basically there is a conflict through more opened editors and better way would be appropriate. In fact as will be seen further the definition of constraints for profiles is done in such a way that this conflict will not cause any issue no matter what DEMO profiles are opened in different editors and how many of them.

Chapter 4

Realisation

4.1 Implementation

Having implemented fixes and extensions for UML Profiles on OpenPonk platform the target implementation of DEMO diagramming is possible. The following steps are basically transforming specified profiles into Pharo code in OpenPonk so it can be used in the class editor to diagramming.

4.1.1 Profile Factories

In order to implement custom profiles there is a factory in OpenPonk to inherit a new factory to generate a custom profile. The name of that default factory is OPUmlCustomProfile. The above specified profiles were defined in newly created factories that were implemented as childs of the class OPUml-CustomProfile, more precisely the main DEMO profile is defined in a factory that is a child of the class OPUmlCustomProfile and the other profiles are defined in factories that are childs of the DEMO profile:

- OPUmlCustomProfile
    - DEMOProfileFactory
        - OCDProfileFactory
        - TPTProfileFactory
        - BCTProfileFactory
        - AOMProfileFactory
        - OFDProfileFactory

So it can be said that the layout of factories reflects the profile layout previously stated.

4.1.1.1 DEMO Profile Factory

The main profile factory is to generate the DEMO profile representing the DEMO metamodel. The output is the realisation of the theoretically defined DEMO UML profile in the Profiles Layout Chapter. The profile was created with the attributes as described in Profile Creation Section. The attribute name was set to 'DEMO'. The implementationPrefix was set to 'D' (stands for DEMO). All the elements the metamodel is consists of were set into the packagedElements. Additionally, relationship constraints of the metamodel were set for the profile.

4.1.1.2 The Concrete Profile Factories

The concrete profile factories corresponds to the target profiles intended to diagramming. The relations of the profile factory to the generated profile and to the theoretically defined UML profile are the same as at the DEMO Profile Factory. The name of the profiles was set to 'DEMO ' + the shortcut of the target diagram (for instance, 'DEMO OCD'). The important settings is the implementationPrefix which needs to be set in the same way as it is set for the DEMO profile factory. This enables use of all the base classes related to the DEMO profile also for the concrete profiles. In other words the objects generated by the main profile factory are shared between all the DEMO profiles. Because the main profile represents the DEMO metamodel, the generated classes covers all the other profiles and no stereotype is defined specifically for any of the concrete profiles. Which is aimed for reasons of use of one common model that can be viewed from different views. The getters of super class – DEMO profile factory – provide the composed elements that are needed to compose the profile; only the subset, defined in Profiles Layout Section for the corresponding profile, of the elements is used. On top of what is described in Profile Creation Section there were added constraints (see the following section) into the profile.

4.1.2 Constraints

Constraints are implemented as a part of the defined profile. It is possible to define constraints separately for each profile, nevertheless taking into account (1) the definition of constraints is relatively small, (2) the organisation of the profile factories – the factories of the concrete profiles are childs of the main DEMO factory - and (3) that having loaded more constraints for a profile than is relevant is not an issue, (→) all the derived constraints are defined together for the main factory OPUmlDEMOFactory and they are taken individually by each factory to create a profile so every profile has elements corresponding to the appropriate diagram and constraints corresponding to the whole DEMO metamodel. Exceptions – not implemented constraints for the metamodel - are the constraints for Process Structure Diagram and Action Rule Specification.

Those are currently omitted; both are specific diagrams and the implementation of them is not part of this thesis, besides ARS will have potentially more objects and it needs to be designed how it will be approached regarding diagramming (relationship between transaction kind + step kind and action rule – so there will be another view on the action rule; or more detailed diagramming where it is possible to connect some kind of elements within the action rule which leads to a definition of constraints on more detailed level).

4.2 Results of simplicity of the solution

The use of the DEMO metamodel on the background and the application of UML profiles ensured not only consistency between all the DEMO diagrams but also:

1. The consistency is guaranteed implicitly. It is based on principle.
2. The design leads to a minimum set of checks, minimum logic of any check, nothing redundant.
3. There is no transfer of data between diagrams. It is possible to change any element and the change is seen in a different diagram, to add an element and it is also in other appropriate diagrams. There are no additional processes when a profile is switched.

Of course, this way there is nothing additional to operate for a user and the process of diagramming is smooth for a diagram as well as between diagrams. No button for any check, adding is without duplicating an entry, no messages to user that something is inconsistent and needs to edit.

Conclusion

Evaluation

The solution was designed and implemented in accordance with the requirements. As required, diagramming for DEMO methodology was implemented using UML profiles on OpenPonk platform. The consistency throughout all the diagrams was ensured.

Further Development

The dimensions of development are as follows:

- Implementation of more complex diagrams where UML profile is not enough to cover needed functionality or another level of representation is needed (Process Structure Diagram, Action Rule Specification) and other diagrams (Derived Fact Specification, Work Instruction Specifications)
- Front-end, graphical representation of the elements in the class editor:
  - Icons, assign icons to the elements as defined in section Profile Extension Mechanism
  - Layer for representation of tables, for instances table for TPT (not specified in UML Profiles)
- Adding attributes related to the stereotype. Those are tagged values from the UML profile point of view. For instance transaction has name and identificator (T1) – one should be added as a tagged value.

Bibliography

Agilevisualization.com. (2019). *Agile Visualization*. [online] Available at: http://agilevisualization.com/ [Accessed 13 Feb. 2019].

Bergel, A., Cassou, D., Ducasse, S. and Laval, J. (2013). *Deep into Pharo*. [Erscheinungsort nicht ermittelbar]: Square Bracket Associates.

Dietz, J. (2015). *The essence of organisation*. 2nd ed.

Fuentes, L. and Vallecillo, A. (2004). An introduction to UML profiles. [online] Available at: https://www.researchgate.net/profile/Antonio_Vallecillo/publication/245346983_An_introduction_to_UML_profiles/links/02e7e537c492be345d000000.pdf [Accessed 13 Feb. 2019].

Object Management Group (2017). Meta-Modeling and the OMG Meta Object Facility. [online] Available at: https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf [Accessed 13 Feb. 2019].

Object Management Group (2017). OMG® Unified Modeling Language® (OMG UML®). [online] Available at: https://www.omg.org/spec/UML/ [Accessed 13 Feb. 2019].

Uhnak, P. (2018). *OpenPonk modeling platform — OpenPonk modeling platform documentation*. [online] Openponk.org. Available at: https://openponk.org/ [Accessed 13 Feb. 2019].

Appendix A

Contents of enclosed CD

```
│ ............................................ assignment of master's thesis
├─ ........................................................... the thesis text
└─ ................................................... the thesis text source
```