Generating Effective Test Suites for Model Transformations Using Classifying Terms

Loli Burgueño[1], Frank Hilken[2], Antonio Vallecillo[1], and Martin Gogolla[2]

[1] Universidad de Málaga. {loli,av}@lcc.uma.es
[2] University of Bremen. {fhilken,gogolla}@informatik.uni-bremen.de

Abstract. Generating sample models for testing a model transformation is no easy task. This paper explores the use of classifying terms and stratified sampling for developing richer test cases for model transformations. Classifying terms are used to define the equivalence classes that characterize the relevant subgroups for the test cases. From each equivalence class of object models, several representative models are chosen depending on the required sample size. We compare our results with test suites developed using random sampling, and conclude that by using an ordered and stratified approach the coverage and effectiveness of the test suite can be significantly improved.

1 Introduction

One of the main problems of existing model transformation testing techniques lies in the difficulty of selecting effective test cases [1]. This is specially important when dealing with large models because they normally include many different aspects of the system under study, which usually deserve individual treatment and particular coverage. This is why homogeneous sampling techniques for selecting object models that constitute the test cases tend to fall short.

In this paper we explore a new technique for developing test cases for models in an orderly and comprehensive manner. The approach uses classifying terms [2] for partitioning the model space into a set of representative model classes and for selecting a set of object models from each equivalence class in order to implement *stratified sampling* [3,4]. Stratified sampling is a mature probability sampling technique wherein the designer divides the entire test case population into different subgroups (or *strata*), each one focusing on particular characteristics of the test models, and then selects proportionally the final samples from the different subgroups.

We compare the effectiveness of our proposal with regard to random sampling, using a case study of a model transformation under test for which we have identified some kinds of models of particular interest for the tester. We generate a set of test models using random sampling and analyse their coverage. Then we show how it is possible to generate specific test models for the reference equivalence classes hence significantly improving the effectiveness and coverage of the samples.

This paper is organized in six sections. After this introduction, Section 2 introduces the preliminary concepts used in the paper: tracts, classifying terms and stratified sampling, as well as the running example used to illustrate our proposal. Then, Section 3

describes the technical details of the extension (w.r.t. [2]) we have developed to classifying terms that permits generating more than one representative model per equivalence class and discusses other statistical aspects of the proposal. Section 4 presents the results of the experiment conducted. Finally, Section 5 relates our work to other similar approaches and Section 6 concludes and outlines some future work.

2 Background

2.1 Tracts

Tracts were introduced in [5] as a specification and black-box testing mechanism for model transformations. They are a particular kind of *model transformation contracts* [1,6] especially well suited for specifying model transformations in a modular and tractable manner. Tracts provide modular pieces of specification, each one focusing on a particular transformation scenario. Thus each model transformation can be specified by means of a set of Tracts, each one covering a specific use case—which is defined in terms of particular input and output models and how they should be related by the transformation. In this way, Tracts permit partitioning the full input space of the transformation into smaller, more focused behavioral units, and to define specific tests for them. Tracts also count on tool support for checking, in a black-box manner, that a given implementation behaves as expected—i.e., it respects the Tracts constraints [7].

The main components of the Tracts approach were described in [5]: the source and target metamodels, the transformation T under test, and the transformation contract, which consists of a Tract *test suite* and a set of Tract constraints. In total, five different kinds of constraints are present: the general constraints defined for the source and target models, and the *source*, *target*, and *source-target* Tract constraints imposed for a given transformation.

If we assume a source model m being an element of the test suite and satisfying the source metamodel and the source Tract constraints given, the Tract essentially requires the result $T(m)$ of applying transformation T to satisfy the target metamodel and the target Tract constraints, and the tuple $\langle m, T(m)\rangle$ to satisfy the source-target Tract constraints.

To illustrate Tracts, consider a simple model transformation, `BiBTex2DocBook`, that converts the information about proceedings of conferences (in `BibTeX` format) into the corresponding information encoded in `DocBook` format[3]. The source and target metamodels that we use for the transformation are shown in Fig. 1. Seven constraint names are also shown in the figure. These constraints are in charge of specifying statements on the source models (e.g., proceedings should have at least one paper; persons should have unique names); and on the target models (e.g., a book should have either an editor or an author, but not both).

In addition to constraints on the source and target models, tracts impose conditions on their relationship—as they are expected to be implemented by the transformation's execution. In this case, the `Tract` class serves to define the source-target constraints for the exemplar tract that we use (although several tracts are normally defined for a

[3] `http://docbook.org/`

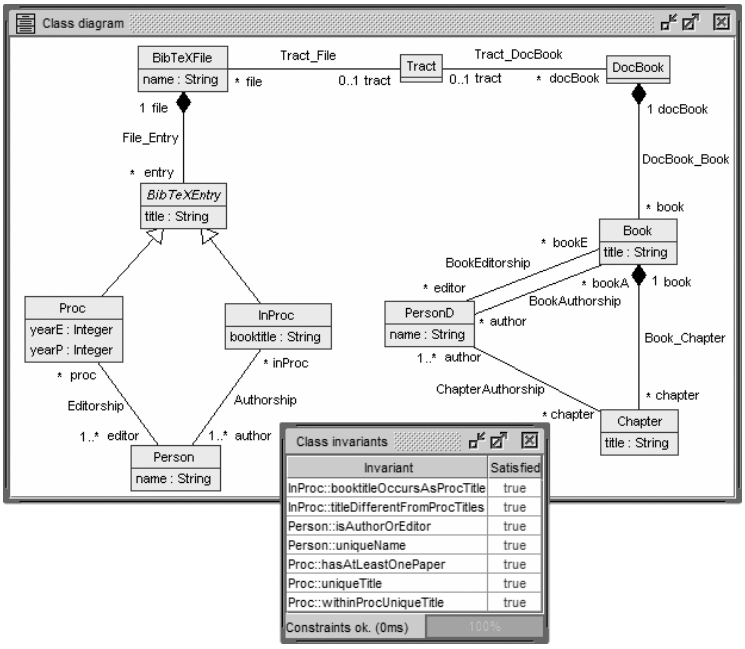

Fig. 1. Source and target metamodels.

transformation, each one focusing on specific aspects or use-cases of the transformation, for simplicity we will consider only one tract here). The following conditions are part of the source-target constraints of the tract:

```
context t:Tract inv sameSizes:
  t.file->size() = t.docBook->size() and
  t.file->forAll( f | t.docBook->exists( db |
    f.entry->selectByType(Proc)->size() = db.book->size()))

context prc:Proc inv sameBooks:
  Book.allInstances->one( bk | prc.title = bk.title and
    prc.editor->forAll(pE | bk.editor->one( bE | pE.name = bE.name )))

context pap:InProc inv sameChaptersInBooks:
  Chapter.allInstances->one( chp |
    pap.title = chp.title and pap.booktitle = chp.book.title and
    pap.author->forAll(aP | chp.author->one(cA | aP.name=cA.name)) )
```

2.2 Tract Test Suites

In addition to the source, target and source-target tract constraints, *test suites* play an essential role in Tracts. Test suite models are pre-defined input sets of different sorts aimed to exercise the transformation. Being able to select particular patterns of source models offers a fine-grained mechanism for specifying the behaviour of the transformation, and allows the model transformation tester to concentrate on specific behaviours of the tract. It also enables test repeatability.

The generation of test suites for tracts has been traditionally achieved by random sampling, i.e., by selecting several object models at random, normally those selected by using different technologies for exploring all possible valid configurations (object models) of a metamodel, such as logic programming and constraint solving [8], relational logic and Alloy [9], term rewriting with Maude [10] or graph grammars [11].

We also proposed the ASSL language (A Snapshot Sequence Language) [12], an imperative language developed to generate object diagrams for a given class diagram. ASSL provides features for randomly choosing attribute values or association ends. Although quite powerful, imperative approaches such as this one are cumbersome to use and also present some limitations in practice [13].

2.3 Determining the sample size

Several statistical works [3,4,14] show how the total sample size is calculated depending on several parameters, which include the size of the entire population (N), the required confidence level ($1 - \alpha$) and the precision (d) of the resulting estimates. For example, one may wish to have the 95% confidence interval be less than 0.06 units wide (i.e., a required precision of 0.03).

Confidence levels are given by their corresponding critical values (also called Z_α-values), which are defined in, e.g., [14]. For example, the Z_α-value for a confidence level of 95% is 1.96 and for a confidence level of 99% is 2.58.

Regarding the size N of the entire population, in our case the population consists of all possible valid models, which is usually a very large and unknown number and then normally considered infinite for statistical purposes [3].

Finally, in this context we are concerned with the estimation of *proportions*, since we can formulate the problem as the estimation of the ratio of models that fail the test, e.g., that are incorrectly transformed by the model transformation. The estimator of that ratio is $p = X/n$, where n is the sample size and X is the number of successes, i.e., "positive" observations (e.g. the number of models for which the transformation fails). If the ratio of positive observations is unknown, the recommended practice is to assume it to be $p = 0.5$.

With all this, the formula that we have to use to determine the sample size n is $n = (Z_\alpha^2 \times p \times (1 - p))/d^2$. The result of the formula is always rounded to the next integer value. Table 1 shows some examples of sample sizes according to some of these parameters.

2.4 Classifying terms

Classifying terms [2] constitute a technique for developing test cases for UML and OCL models, based on an approach that automatically constructs object models for class

Table 1. Examples of sample sizes according to different parameter values.

Confidence Level	95%	99%	95%	99%	95%	99%	95%	99%
Failure rate (unkown=50%)	5%	5%	5%	5%	50%	50%	50%	50%
Precision (uncertainty)	5%	5%	3%	3%	5%	5%	3%	3%
Resulting sample size	**73**	**126**	**203**	**351**	**384**	**666**	**1,067**	**1,849**

models enriched by OCL constraints. By guiding the construction process through so-called classifying terms, the built test cases in form of object models are classified into equivalence classes.

Classifying terms are arbitrary OCL expressions on a class model that calculate a characteristic value for each object model. The expressions can either be boolean, allowing to define up to two equivalence classes; or numerical, where each resulting number defines one equivalence class. Each equivalence class is then defined by the set of object models with identical characteristic values and by one canonical representative object model.

The process to build the test suite is straightforward. We begin by identifying the *sorts* of models that we would like to be part of the test suite. Each sort is specified by a classifying term, that represents the equivalence class with all models that are *equivalent* according to that class, i.e., which belong to the same sort. Once the classifying terms are defined for a tract, the USE tool generates one representative model for each equivalence class. These *canonical* models constitute the test suite of the tract.

For example, suppose that we want to concentrate on different characteristics of the input models of the `BibTex2DocBook` transformation. First, proceedings have two dates: the year in which the conference event was held (`yearE`) and the year in which the proceedings were published (`yearP`). We want to have input models in which these two dates coincide in at least one proceeding, and other input models for which the conference event and publication years are always different. Second, we want to have some sample input model in which one person is editor of at least one proceeding and also author of at least one inproceeding, and also models in which the persons are either author or editors, but not both. Finally, we want to have some source models with at least one inproceeding whose title coincides with its booktitle, and other input models where the titles and booktitles of inproceedings are always different.

As mentioned above, producing test suite models to cover all these circumstances by an imperative approach or by ASSL is normally tedious and error prone. However, the use of classifying terms greatly simplifies this task. It is enough to give three Boolean terms to the model validator, each one defining the classifying term that specifies the characteristic we want to identify in the model. In this case, these Boolean terms are the ones shown below.

```
[ yearE_EQ_yearP ]
    Proc.allInstances->exists(yearE=yearP)

[ twoRolePerson ]
    Person.allInstances->exists (proc->size>0 and inProc->size>0)

[ title_EQ_booktitle ]
    BibTeXEntry.allInstances->exists(be |
        InProc.allInstances->exists(ip | be.title=ip.booktitle))
```

Given that an object model either belongs or not to any of the equivalence classes, the three classifying terms define eight equivalence classes ($8 = 2^3$) and the model validator selects one representative for each one of them by scrolling through all valid object models. For this, as described in [2], each classifying term is assigned an integer value, and the values of the classifying terms are stored for each solution. Using the classifying terms and these values, constraints are created and given to a Kodkod solver

along with the class model during the validation process. Both the USE tool and the model validator plugin are available for download from `http://sourceforge.net/projects/useocl/`.

3 Extending Classifying Terms

3.1 Selecting more than One Sample per Classifying Term

Classifying terms permit checking that the behaviour of the transformation is as expected for the selected sample models. However, this does not prove that the transformation will always work. For instance, the transformation might behave differently if the model validator selects other representative models.

This is why it would be interesting to ask the model validator to produce more than one model for each equivalence class. There is another good reason for that: we know that not all sorts of input models have the same likelihood of happening in reality. Thus, we can select more sample models for those equivalence classes that we think are more frequent. In this way we can exercise the model transformation in a more focused manner, and produce a richer test suite for the tract (and hence for the transformation).

We have extended the model validator to permit the creation of a given number of representative object models per equivalence class. Then, instead of stopping after one representative object model is found for a class, the model validator continues searching until that number is reached or the exploration finishes. The question now is to determine how many object models of each class should be generated, and this is where the *stratified sampling* statistical technique comes into play.

3.2 Stratified sampling

Sampling is a mature statistical technique aimed at selecting a subgroup of a population (i.e., a sample) to estimate some characteristics of the whole population. In our context, the population is the set of all possible object models that conform to a metamodel, and our goal is to select a representative sample from within them that can be used for testing an operation on the model. For example, to serve as input for a model transformation.

Sampling theories and methods are nowadays well known. They have become essential tools used by politicians and marketing managers in their polls and surveys to estimate the intentions of a target population. We all know that they are able to obtain very precise and accurate results with rather small samples, and in record time [3,4,14]. In computer graphics they are very useful for, e.g., rendering 3D models [15]), and in software engineering they have been applied for the estimation of software reliability [16,17] and also for fault injections [18].

Stratified sampling permits considering the population organized into different subgroups (strata), each one with particular characteristics and relevance for the study. Each stratum is then sampled as an independent population, out of which individual elements can be randomly selected.

There are several potential benefits to stratified sampling. First, dividing the population into distinct, independent strata enables researchers to draw inferences about

specific subgroups that may be lost in a more generalized random sample. Second, utilizing a stratified sampling method can lead to more efficient statistical estimates, provided that strata are selected based upon relevance to the criterion in question. Even if a stratified sampling approach does not lead to increased statistical efficiency, such a tactic will not result in less efficiency than would simple random sampling, provided that each stratum is proportional to the group's size in the population. Finally, since each stratum is treated as an independent population, different sampling approaches can be applied to different strata, potentially enabling researchers to use the approach best suited (or most cost-effective) for each identified subgroup within the population [19].

A stratified sampling approach is most effective when three conditions are met: variability within strata is minimized; variability between strata is maximized, and the variables upon which the population is stratified are strongly correlated with the desired dependent variable [14]. However, stratified sampling also presents some limitations, the difficulty of the specification of the different strata and the characterization of their members being the most significant ones. But this is precisely where classifying terms are strong, since they provide a declarative approach for specifying the equivalence classes that are of interest to the model transformation tester, and that constitute a partition of the source model space.

3.3 Determining the sample size for each stratum

Once we know the complete sample size (Section 2.3), we need to decide the number of elements of each stratum (equivalence class). For this we propose the use of *proportional allocation*, whereby the sample size of each stratum respects the proportion of elements that each stratum represents from the entire population. Hence, the more elements belong to a group, the larger the sample size for the stratum representing that group.

In our case, we need to know (or estimate) the proportion of each stratum. One usual method is to use some a priori knowledge about the population to estimate these proportions [4]. There are also several techniques that can be applied in case we do not have any prior knowledge or we do not want to guess, as described in [4]. In our example, where we had 8 equivalence classes corresponding to the 3 classifying terms specified for the `BibTeX` metamodel, we have subjectively estimated that the chances of a model to fulfil each of the classifying terms (`yearE_EQ_yearP`, `twoRolePerson` and `title_EQ_booktitle`) are, respectively, 90%, 70% and 30%. To be more precise, this estimation could be done in the future using a database like DBLP[4].

Table 2 lists these classes and the proportions resulting according to the estimations we have made. It also shows the number of elements (object models) of each stratum depending on the total sample size (as previously described in Table 1). For example, if the total sample size is 203 the corresponding sample sizes for each stratum are, respectively, 39, 90, 17, 39, 5, 10, 2 and 5 (they add 207 because all numbers are always rounded up).

[4] http://dblp.uni-trier.de/

Table 2. Equivalence classes and their estimated proportions.

	yearE_EQ_yearP	twoRolePerson	title_EQ_booktitle	Proportion	**76**	**130**	**207**	**356**	**389**	**667**	**1,071**	**1,852**
1	true	true	true	19%	14	24	39	67	73	126	202	350
2	true	true	–	44%	33	56	90	155	170	294	471	816
3	true	–	true	8%	6	11	17	29	32	54	87	150
4	true	–	–	19%	14	24	39	67	73	126	202	350
5	–	true	true	2%	2	3	5	8	9	14	23	39
6	–	true	–	5%	4	7	10	18	19	33	53	91
7	–	–	true	1%	1	2	2	4	4	6	10	17
8	–	–	–	2%	2	3	5	8	9	14	23	39

4 Experimental Results

We checked the effectiveness of our approach by first generating some test suites using random sampling, and then by comparing the samples with the ones obtained using our approach. We analyzed how well the random samples cover the equivalence classes defined by the model tester, calculating the percentage of the generated object models that belonged to each class (see Table 2). Random sampling was performed by asking the model validator to generate these numbers of object models by exploring all possible valid configurations (object models) of the source metamodel and returning those that satisfy the constraints.

Table 3 shows the results of the experiment. The fifth column shows the expected percentage of object models that should fall into each equivalence class, while the last eight columns show, for each of the selected sample sizes, the actual percentage of object models from those generated by the model validator. For readability reasons, the numbers have been rounded to the closest integer.

Table 3. Coverage by object models generated using random sampling.

				Expected	Actual coverage							
	yearE_EQ_yearP	twoRolePerson	title_EQ_booktitle	Proportion	**76**	**130**	**207**	**356**	**389**	**667**	**1,071**	**1,852**
1	true	true	true	**19%**	11%	13%	9%	5%	5%	3%	4%	4%
2	true	true	–	**44%**	88%	68%	53%	41%	40%	37%	44%	38%
3	true	–	true	**8%**	0%	0%	0%	0%	0%	0%	0%	0%
4	true	–	–	**19%**	0%	0%	1%	7%	7%	5%	3%	5%
5	–	true	true	**2%**	1%	1%	1%	0%	0%	0%	1%	2%
6	–	true	–	**5%**	0%	18%	36%	35%	34%	44%	41%	42%
7	–	–	true	**1%**	0%	0%	0%	0%	0%	0%	0%	0%
8	–	–	–	**2%**	0%	0%	0%	12%	14%	11%	7%	9%

As expected, we observe that using random sampling the coverage of the input domain is quite limited. The models generated belong only to some equivalence classes and leave the rest of them with no representation. For our first sample size, only classes 1, 2 and 5 have representatives. We need a sample size of 207 to have object models in 5 of the 8 classes, and a sample size of $1,071$ to have them in 6 of the 8 classes. Even for our larger sample, in which $1,852$ object models are generated, the equivalence classes 3 and 7 have no representation. Note that these classes are not empty because

they are inconsistent—representatives were generated for them using classifying terms and stratified sampling.

Furthermore, the distribution of the representatives among the equivalence classes does not correspond to the expected. For instance, for our larger sample size, class 6 is not very relevant thus, only 5% of the object models are expected to represent it. Nevertheless, using random sampling, 42% of the object models belong to it.

5 Related Work

Several approaches exist to generate object models from class models using different languages and tools. The USE model validator, used in this work, is based on the transformation of UML and OCL into relational logic [20]. Another approach within the same tool, USE, is based on the language ASSL (A Snapshot Sequence Language) [12] that we have discussed above. Its imperative nature requires more effort by the developer and makes the specification of the test object models a more complex and cumbersome task than the declarative approach described here. We have also mentioned other approaches to generate object models relying on different technologies like logic programming and constraint solving [8], relational logic and Alloy [9], term rewriting with Maude [10] or graph grammars [11]. However, they do not support full OCL and only provide random sampling generation.

Finally, there is a vast amount of literature about sampling techniques. Among them, stratified sampling provides interesting benefits when the population is not homogeneous, being more effective than random sampling [19]. Stratified sampling is commonly used nowadays in quality assurance [21], surveys and polls [14]. In the Informatics domain, it has proved to be very effective in computer graphics for representing 3D models [15], wherein the sampling strategy is critical for point rendering systems which rely exclusively on sampled geometry to represent models [22]. In software engineering, it has been mainly applied for the estimation of software reliability [16,17] and also for fault injections [18]. However, we are not aware of any other work in the field of model-based engineering that uses stratified sampling to generate test object models, as we propose here.

6 Conclusions

This paper proposes a new technique for developing test cases for model transformation testing. It uses stratified sampling, a statistical technique for selecting a small subgroup of a population that can be representative enough to estimate characteristics of the population, or to predict its behaviour. We have illustrated with one example how this technique can be used, and how it can be more effective than random sampling.

This initial work can be continued in various directions. In particular, we would like to conduct more exhaustive tests with different kinds of model transformations and with other classifying terms, in order to get a better estimation of the effectiveness, strengths and limitations of our proposal. Larger case studies should also give more feedback on the features and scalability of the approach. Finally, we are working on the integration of this work into an automated model transformation testing environment that would fully support tracts.

References

1. Baudry, B., Dinh-Trong, T., Mottu, J., Simmonds, D., France, R., Ghosh, S., Fleurey, F., Le Traon, Y.: Model transformation testing challenges. In: ECMDA WS. on Integration of MDD and Model Driven Testing. (2006)
2. Gogolla, M., Vallecillo, A., Burgueño, L., Hilken, F.: Employing classifying terms for testing model transformations. In: Proc. of MODELS'15, IEEE (2015) 312–321
3. Särndal, C.E., Swensson, B., Wretman, J.: Model Assisted Survey Sampling. Springer (2003)
4. Thompson, S.K.: Sampling. John Wiley & Sons (2012)
5. Gogolla, M., Vallecillo, A.: *Tract*able model transformation testing. In: Proc. of ECMFA'11. Volume 6698 of LNCS., Springer (2011) 221–236
6. Cariou, E., Marvie, R., Seinturier, L., Duchien, L.: OCL for the specification of model transformation contracts. In: Proc. of the OCL and Model Driven Engineering Workshop. (2004)
7. Burgueño, L., Wimmer, M., Troya, J., Vallecillo, A.: Static Fault Localization in Model Transformations. IEEE Transactions on Software Engineering **41**(5) (2015) 490–506
8. Cabot, J., Clarisó, R., Riera, D.: UMLtoCSP: A Tool for the Formal Verification of UML/OCL Models using Constraint Programming. In: Proc. of ASE'07, ACM (2007) 547–548
9. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: On Challenges of Model Transformation from UML to Alloy. Software and System Modeling **9**(1) (2010) 69–86
10. Roldán, M., Durán, F.: Dynamic Validation of OCL Constraints with mOdCL. ECEASST **44** (2011)
11. Ehrig, K., Küster, J.M., Taentzer, G.: Generating instance models from meta models. SoSyM **8** (2009) 479–500
12. Gogolla, M., Bohling, J., Richters, M.: Validating UML and OCL Models in USE by Automatic Snapshot Generation. Software and Systems Modeling **4**(4) (2005) 386–398
13. Offutt, J., Irvine, A.: Testing object-oriented software using the category-partition method. In: Proc. of TOOLS USA'95. (1995) 293–304
14. Groves, R.M., Fowler Jr., F.J., Couper, M.P., Lepkowski, J.M., Singer, E., Tourangeau, R.: Survey Methodology. 2 edn. Wiley (2009)
15. Nehab, D., Shilane, P.: Stratified point sampling of 3D models. In: Proc. of SPBG'04, Eurographics Association (2004) 49–56
16. Li, Q.Y., Luo, L.: Minimum test case for discrete-type software reliability testing by stratified sampling. Applied Mechanics and Materials **1874** (2012) 263–266
17. Podgurski, A., Masri, W., McCleese, Y., Wolff, F.G., Yang, C.: Estimation of software reliability by stratified sampling. ACM Trans. Softw. Eng. Methodol. **8**(3) (1999) 263–283
18. de Oliveira Moraes, R.L., Martins, E., Poletti, E.C.C., Mendes, N.V.: Using stratified sampling for fault injection. In: Proc. of LADC'05. Volume 3747 of LNCS., Springer (2005) 9–19
19. Nair, V.N., James, D.A., Ehrlich, W.K., Zevallos, J.: A statistical assessment of some software testing strategies and application of experimental design techniques. Statistica Sinica **8** (1998) 165–184
20. Kuhlmann, M., Gogolla, M.: From UML and OCL to relational logic and back. In: Proc. of MODELS'12. Volume 7590 of LNCS., Springer (2012) 415–431
21. Galin, D.: Software Quality Assurance: From Theory to Implementation. Pearson Education (2004)
22. Grossman, J.P., Dally, W.J.: Point sample rendering. In: Proc. of Rendering Techniques'98, Springer (1998) 181–192