The Class Responsibility Assignment Case

Martin Fleck
Business Informatics Group
TU Wien, Austria
fleck@big.tuwien.ac.at

Javier Troya
ISA Research Group
Universidad de Sevilla, Spain
jtroya@us.es

Manuel Wimmer
Business Informatics Group
TU Wien, Austria
wimmer@big.tuwien.ac.at

Abstract

This paper describes a case study for the ninth Transformation Tool Contest (TTC'16)[1]. The case is aimed at the production of high-quality designs for object-oriented systems and presents the problem of finding a good class diagram for a given set of methods and attributes with functional and data relationships among them. In order to obtain such a class diagram, dedicated quality metrics that have been defined in the context of the class responsibility assignment problem need to be optimized. Therefore, the focus of this case study is not on the definition of the necessary set of rules, but rather on the orchestration of such rules in order to find the optimal class diagrams. The evaluation of the produced transformation is driven by the quality of the produced models, the complexity of the rule orchestration as well as by the flexibility of the solution and its performance.

1 Introduction

The quality of an object-oriented design has a direct impact on the quality of the code produced: the higher the quality of the models in the design, the higher the quality of the code. The Class Responsibility Assignment (CRA) problem [BBL10] deals with the creation of such high-quality object-oriented models. When solving the CRA, you need to decide where responsibilities, under the form of class operations and attributes they manipulate, belong and how objects should interact. Therefore, CRA is a non-trivial problem, often requiring human judgement and decision-making [MJ14]. The necessity to assign responsibility to classes can arise in the context of two problems:

- *Finding a class diagram.* When migrating an application from a procedural language such as C to an object-oriented language, one needs to group similar procedures and their associated variables to create an object-oriented design.
- *Optimizing a class diagram.* During the refactoring of an existing object-oriented application, one may want to reorganize the existing class structure in order to increase quality aspects such as maintainability or readability.

The relevance of this case study for TTC'16 stands out in that it presents a type of problem not presented in any contest before. In the case, the quality of the produced transformations is not strictly measured by the quality of the rules, but by their application orchestration, i.e., to execute the rules in the appropriate order and for the most appropriate matches to obtain high-quality output models. This implies to deal with a high computational complexity, since there are many different possibilities for orchestrating a set of rules. Indeed, we can categorize the CRA problem as a problem related

Copyright © by the paper's authors. Copying permitted for private and academic purposes.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

[1]All artifacts related to this case study can be found on GitHub at:
`https://github.com/martin-fleck/cra-ttc2016`

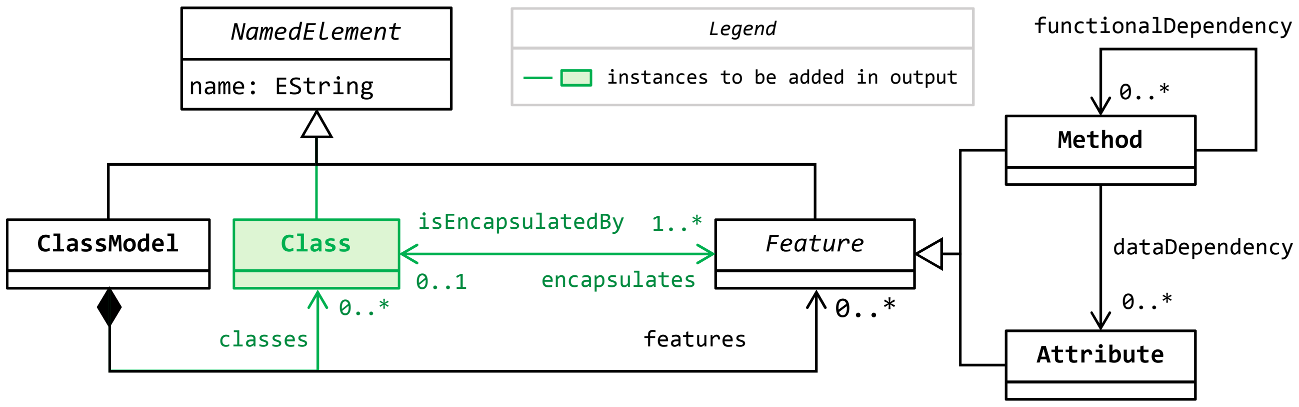


Figure 1: Class Model metamodel

to the partitioning of a set of labeled features (operations and attributes) into non-empty classes so that every feature is included in exactly one class. The number of possible partitions, i.e., classes, is given by the Bell number (cf. Equation 1). The nth of these numbers, B_n, counts the number of different ways a given a set of n features can be divided into classes. If there are no features given (B_0), we can in theory produce exactly one partition (the empty set, $\emptyset$). The order of the classes as well as the order of the features within a class does not need to be considered as the semantic of a class diagram does not depend on that order.

$$\begin{aligned} B_{n+1} &= \sum_{k=0}^{n} \binom{n}{k} B_k \\ B_0 &= 1 \end{aligned} \tag{1}$$

Considering the first Bell numbers, which are shown below (cf. sequence A000110[2] in the OEIS online database for integer sequences), we can see that the number of partition possibilities grows exponentially and is already quite high for a low number of features. For example, an instance where you need to assign 15 features to an unknown amount of classes already yields 1382958545 different possibilities.

```
1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437,
190899322, 1382958545, 10480142147, 82864869804, 682076806159, 5832742205057,
51724158235372, 474869816156751, 4506715738447323, 44152005855084346,
445958869294805289, 4638590332229999353, 49631246523618756274, ...
```

In order to solve the case, described in the next section, several techniques may be applied by the contestants.

2 Case Description

In this case study we propose a simplified version of the CRA problem. Contestants are given a set of methods and attributes as well as dependencies between them. Such a structure is also referred to as responsibilities dependency graph (RDG). Based on the RDG, the goal is to generate a high-quality class diagram (CD) model. The purpose is therefore to create a RDG2CD model transformation, where the RDG must evolve into a CD, categorized as an endogenous model transformation [MVG06], since both the input and output models conform to the same metamodel.

Figure 1 depicts the common metamodel that is used to represent both, the RDG and the output CD. The RDG is the subset of the metamodel containing only the features and their dependencies, and is represented in black, while the additional class and relationships needed to produce a CD are represented in green. The concepts depicted in the metamodel are summarized as follows:

Class Classes represent classes as known from object-oriented programming and modeling languages. A class hereby encapsulates certain functionality aspects in terms of methods, which in turn use data stored in attributes of instances of the same or other classes. In this sense, classes serve as a container object for behavioral features (*methods*) and data features (*attributes*).

[2]http://oeis.org/A000110

Table 1: Summary of input models

	Input A	Input B	Input C	Input D	Input E
Attributes	5	10	20	40	80
Methods	4	8	15	40	80
Data Dep.	8	15	50	150	300
Functional Dep.	6	15	50	150	300

Method A method is used to describe a piece of executable behavior relating to a certain functionality. In the given metamodel, we abstract from the behavioral details, such as the sequence of performed actions, and focus on the dependencies generated through those actions. Specifically, we distinguish between *functional dependencies*, which refer to the dependencies among methods, and *data dependencies*, which refer to the dependencies between methods and attributes.

Attribute Attributes are used to store concrete data values. When used in an object-oriented language, all attributes of an object with their associated values represent the current state of the object. Changes of that state are typically executed through methods calls.

Encapsulation The encapsulation of classes and their features, i.e., methods and attributes, is given in the form of a bidirectional association (associations *isEncapsulatedBy* and *encapsulates*).

Functional Dependency A functional dependency represents a uni-directional relation from one method to another method, for example a method call. A functional dependency can therefore be defined as a relation $f \in M \times M$, where M is the set of all provided methods.

Data Dependency A data dependency represents a uni-directional relation from one method to an attribute, for example a read or write access. A data dependency can therefore be defined as a relation $d \in M \times A$, where M is the set of of all provided methods and A is the set of all provided attributes.

2.1 Input

As input, contestants are given the above specified metamodel in Ecore as well as a set of input models conforming to the RDG subset. All input models are provided as XMI files and it can be assumed that the names of the methods and attributes contained in one RDG are unique. In total there are five input models with a varying degree of complexity. A summary of those models is given in Table 1.

2.2 Transformation

In this section we introduce the two main aspects that have to be considered by the contestants when producing output models.

2.2.1 Transformation Rules.

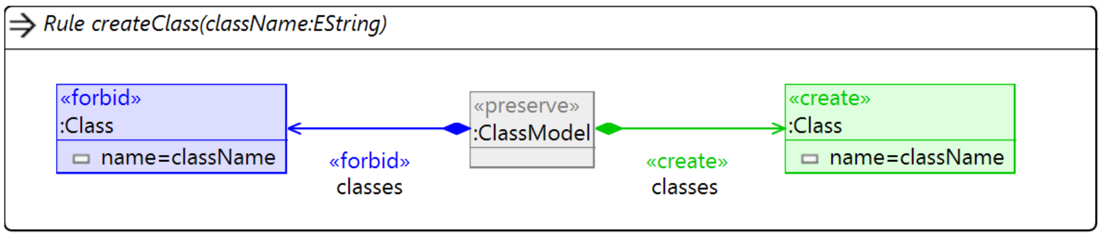
As mentioned previously, the goal of this case study is to generate a high-quality CD model from the given RDG input model. To generate such a CD model, two major tasks have to be performed for each input model:

1. Create an appropriate amount of class instances, each having a unique name (the names do not have to be meaningful and can be assigned randomly).
2. Assign features to classes, i.e., set the encapsulation relationship between the classes and all the given features.

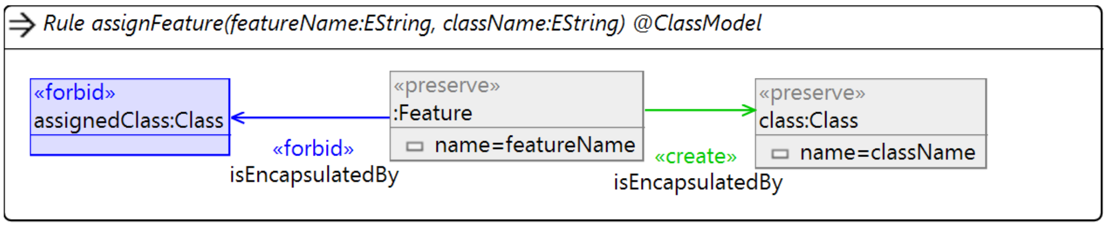
In order to create classes and assign features, two simple rules realizing the above mentioned tasks may be used. An example implementation of these rules in Henshin [ABJ^{+}10] is depicted in Figure 2.

2.2.2 Transformation Goals.

The true challenge in this case study does not lie in the correct implementation of the transformation rules and creation of conforming CD models, but rather in the creation of high-quality CD models. Producing a class diagram where the right number of classes is chosen and a proper assignment of features is realized is a non-trivial task, and the contestants may realize it in several different ways.



(a) *createClass* rule.



(b) *assignFeature* rule.

Figure 2: Implementation of the rules in Henshin.

To begin with, they may define more rules other than the two shown in Figure 2, or they can extend the given rules. Contestants also need to find the appropriate orchestration of the rules, with the appropriate input parameters, for producing high-quality class diagrams. Since this is quite complex and the state space can be huge (depending on the size of the input models), contestants may use several techniques for finding the rules orchestration, such as backtracking, state space exploration or dedicated search-based techniques.

Please note that we do not enforce any particular direction how the search for high-quality CD models is implemented. It may be done by exploiting current features of model transformation engines[3], e.g., used for formal analysis of models by using model checking techniques, by introducing new search features to the model transformation engines [AVS$^+$14], by implementing search algorithms as transformations [DJVV14], combining search frameworks with model transformation engines [FTW15] or by translating the artifacts to dedicated encodings which already provide sophisticated search capabilities [EWZ14]. As several approaches for searching transformation spaces for good solutions have been proposed in the last years, we see this TTC case as an opportunity to evaluate and compare these diverse approaches based on a common example.

As for deciding on the quality of the obtained class diagrams, we use two common metrics for considering the quality of a grouping mechanism such as the grouping of functionality into classes: coupling and cohesion [BBL10]. *Coupling* refers to the number of external dependencies a specific group has, whereas *cohesion* refers to the dependencies within one group. Typically, low coupling is preferred as this indicates that a group covers separate functionality aspects of a system, improving the maintainability, readability and testability of the overall system [YC79]. On the contrary, the cohesion within one group should be maximized to ensure that it does not contain parts that are not part of its functionality. Mapping these definitions to our problem, we can calculate coupling and cohesion as the sum of external and internal dependencies, respectively.

One metric that combines coupling and cohesion into a single quality metric is the so-called *CRA-Index* [MJ14]. To be precise, the CRA-Index uses the coupling and cohesion ratios, i.e., the coupling and cohesion achieved considering the number of classes and attributes, and subtracts the former from the latter. Considering the meaning of coupling and cohesion as explained before, we can conclude that a higher CRA-Index relates to a higher quality of the class diagram. The formulae to calculate all necessary metrics and values are given below (taken from [MJ14]). Please note that $M(c)$ and $A(c)$ refer to all methods and attributes of class c, respectively.

[3]https://wiki.eclipse.org/Henshin_State_Space_Tools

$$
\begin{aligned}
CRA\text{-}Index &= CohesionRation - CouplingRatio\\
CohesionRatio &= \sum_{c_i \in Classes} \frac{MAI(c_i,c_i)}{|M(c_i)| \times |A(c_i)|} + \frac{MMI(c_i,c_i)}{|M(c_i)| \times |M(c_i)-1|}\\
CouplingRatio &= \sum_{\substack{c_i,c_j \in Classes\\ c_i \neq c_j}} \frac{MAI(c_i,c_j)}{|M(c_i)| \times |A(c_j)|} + \frac{MMI(c_i,c_j)}{|M(c_i)| \times |M(c_j)-1|}\\
MMI(c_i,c_j) &= \sum_{\substack{m_i \in M(c_i)\\ m_j \in M(c_j)}} DMA(m_i,m_j)\\
MAI(c_i,c_j) &= \sum_{\substack{m_i \in M(c_i)\\ a_j \in A(c_j)}} DMA(m_i,a_j)\\
DMA(m_i,a_j) &= \begin{cases} 1 & \text{if there is a dependency between method } m_i \text{ and attribute } a_j\\ 0 & \text{otherwise}\end{cases}\\
DMM(m_i,m_j) &= \begin{cases} 1 & \text{if there is a dependency between method } m_i \text{ and } m_j\\ 0 & \text{otherwise}\end{cases}
\end{aligned}
$$

In order to avoid division by zero in the calculation of the cohesion and coupling ratios, zero is assigned to the result of a division whenever its denominator is zero.

To sum up, the challenge of this case study is to find a way to properly orchestrate the given, extended, or newly created rules in order to optimize the quality of the produced class diagrams, which is achieved by optimizing the explained CRA-Index.

2.3 Output

In order to ensure validity of the generated output CD models, they must conform to the metamodel shown in Figure 1 and in addition must satisfy the following constraints:

- Every class must have a unique name, represented in OCL as:

```
-- unique class names
Class.allInstances()->isUnique(name)
```

- All features provided in the input model must be encapsulated by a class, represented in OCL as:

```
-- all features assigned
Feature.allInstances()->forAll(f | not f.isEncapsulatedBy.oclIsUndefined())
```

- There cannot be any empty classes. This is already enforced by a lower bound constraint in the metamodel.

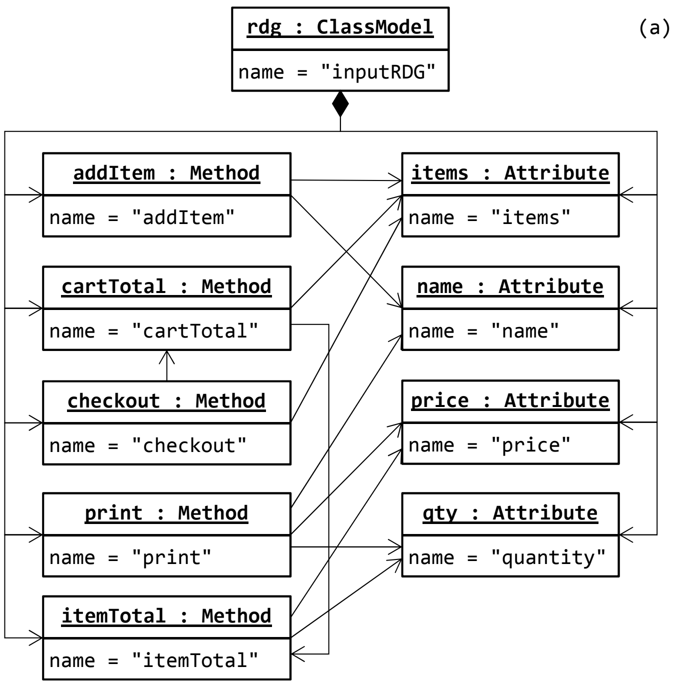
Figure 3(a) depicts an example input RDG model composed of methods, attributes, and the relationships between them. Please note that in this figure we have not labelled the relationships for the sake of presentation. One possible (not necessarily optimal) output class diagram for this RDG is shown in Figure 3(b) in a simplified class diagram-like notation. The actual output CDs that need to be produced by the contestants must be in XMI format. Finally, the metrics obtained for the shown class diagram are depicted in the table in Figure 3(c).

3 Evaluation Criteria

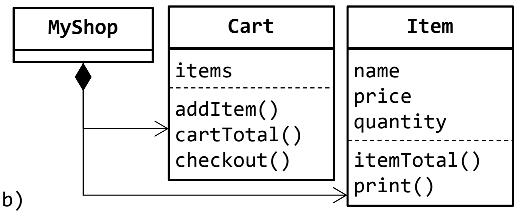
For evaluating the provided solutions to the case study described in this paper, we are interested in the following five criteria. Both their description as well as the way to evaluate them can also be found in the provided evaluation spreadsheet.

Completeness & Correctness Completeness is a criteria indicating whether the provided solution/program always yields an output model as a result for the provided input models. On the other hand, correctness defines whether the generated output model conforms to the output metamodel and fulfils all constraints specified in Section 2.3. Specifically, correctness and completeness count how many of the five provided input models have been transformed into correct output models.

(a)



(b)



(c)

	Cart	Item
MAI(c_i, c_i)	3	5
MMI(c_i, c_i)	1	0
CohesionRatio	1.1667	0.8333
MAI(c_i, c_j)	1	0
MMI(c_i, c_j)	1	0
CouplingRatio	0.4444	0
∑ CohesionRatio	2	
∑ CouplingRatio	0.4444	
CRA-Index	1.5556	

Figure 3: Example input/output model pair with quality characteristics: (a) input model in abstract syntax, (b) output model in concrete syntax, (c) measures for output model.

Optimality With optimality the quality of the correctly generated models is evaluated, i.e., the CRA-Index of the output model. The higher the CRA-Index the better the quality of the output model. Here, the reviewers need to rank the solutions in relation to the other solutions provided and give them points on a scale between 1 and 10, where 1 refers to the worst (possible) solution and 10 refers to the best solution. To support this ranking, we provide the CRA-Index of our reference solution in the evaluation spreadsheet and a program that calculates the CRA-Index for a given class diagram.

Complexity With complexity we measure the efforts needed to provide search capabilities for good solutions as well as to evaluate the solutions based on the given metrics. For instance, this involves to evaluate how much effort has been invested to augment the provided rules, develop orchestration specifications such as providing an explicit control flow for the rules, implement search algorithms as transformations, or to implement transformations to dedicated encodings used for performing the search and back. Here, again, the reviewers need to rank the solutions in relation to the other solutions provided and give them points on a scale between 1 and 10, where 1 refers to the worst (possible) solution and 10 refers to the best solution.

Flexibility Flexibility measures how easy it is to modify the given solution to support additional/other quality metrics besides coupling, cohesion and the CRA-Index. For this criteria, reviewers need to estimate the effort it takes to integrate new objectives (such as fixing the number of classes to a given value) and give the provided solutions points on a scale between 1 and 10, where 1 refers to the worst (possible) solution, i.e., the solution where the most effort is needed, and 10 refers to the best solution, i.e., the integration can be done quickly.

Performance The performance evaluation consists of the measured execution time, i.e., the time it takes the provided solution to generate a high-quality output model for a given input model. Please note that reading the input model and writing the output model is not considered to be part of this performance evaluation. For Java-based solutions, we suggest using Java's internal time measurements, i.e., the method `java.lang.System.nanoTime()`, which is also used by the Apache Commons Lang's[4] `StopWatch` class. All performance values must be given exact to the millisecond, e.g., `03:02.426` meaning 3 minutes, 2 seconds and 426 milliseconds or in total `182426` milliseconds.

All criteria, except the complexity and flexibility of the solution, are evaluated separately on all provided input models.

[4] `https://commons.apache.org/proper/commons-lang/`

Table 2: Evaluation Schema

Criteria	**Weight**	**MaxPoints**	**Total**
Completeness & Correctness	1	10	10
Optimality	2	10	20
Complexity	2.5	10	25
Flexibility	2.5	10	25
Performance	2	10	20
Total			100

Depending on the criteria, each input model is given a specific weight that relates to the complexity of the model. Furthermore, the criteria are also weighted according to their importance. The maximum points and the weight of each criteria is specified in Table 2.

Using this table, which is incorporated in the provided spreadsheet, the result value, i.e., the *final score* of all provided solutions can be calculated. Each solution will receive a score between 0 and 100 and can therefore be ranked in comparison with other solutions.

4 Test artifacts

As a main test artefact, we provide an Excel sheet which needs to be filled out by the reviewers of the solutions. To support the reviewer in this task we provide a small Java application that checks the correctness and completeness of the provided output models and calculates the CRA-Index. To execute the application, we need to call the program with the respective class model:

```
java —jar CRAIndexCalculator.jar <xmi—model>
```

An example of the output of the application is depicted in Listing 1 and Listing 2.

Listing 1: Example output when providing a valid class model

```
java —jar CRAIndexCalculator.jar Cart_Item_cd.xmi

——————————————————————————————————————————
General Info
——————————————————————————————————————————
|Classes|    = 2
|Methods|    = 5
|Attributes| = 4
——————————————————————————————————————————

——————————————————————————————————————————
Correctness
——————————————————————————————————————————
All classes have different names: ok
All features are encapsulated in a class: ok
Correctness: ok
——————————————————————————————————————————

——————————————————————————————————————————
Optimality
——————————————————————————————————————————
The aggregated cohesion ratio is: 2.0
The aggregated coupling ratio is: 0.4444
This makes a CRA—Index of: 1.5555
——————————————————————————————————————————
```

Listing 2: Example output when providing an RDG model

```
java -jar CRAIndexCalculator.jar Cart_Item_rdg.xmi

------------------------------------------
General Info
------------------------------------------
|Classes|    = 0
|Methods|    = 5
|Attributes| = 4
------------------------------------------

------------------------------------------
Correctness
------------------------------------------
All classes have different names: ok
Constraint violated! Feature addItem is not encapsulated in a class
Constraint violated! Feature cartTotal is not encapsulated in a class
Constraint violated! Feature checkOut is not encapsulated in a class
Constraint violated! Feature print is not encapsulated in a class
Constraint violated! Feature itemTotal is not encapsulated in a class
Constraint violated! Feature items is not encapsulated in a class
Constraint violated! Feature name is not encapsulated in a class
Constraint violated! Feature price is not encapsulated in a class
Constraint violated! Feature qty is not encapsulated in a class
Correctness: Violations found.
------------------------------------------

------------------------------------------
Optimality
------------------------------------------
The aggregated cohesion ratio is: 0.0
The aggregated coupling ratio is: 0.0
This makes a CRA-Index of: 0.0
------------------------------------------
```

References

[ABJ+10] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *Proceedings of 13th International Conference on Model Driven Engineering Languages and Systems (MODELS'10)*, volume 6394 of *LNCS*, pages 121–135. Springer, 2010.

[AVS+14] Hani Abdeen, Dániel Varró, Houari A. Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. Multi-objective optimization in rule-based design space exploration. In *Proceedings of the 29th International Conference on Automated Software Engineering (ASE'14)*, pages 289–300. IEEE/ACM, 2014.

[BBL10] M. Bowman, L.C. Briand, and Y. Labiche. Solving the Class Responsibility Assignment Problem in Object-Oriented Analysis with Multi-Objective Genetic Algorithms. *IEEE Transactions on Software Engineering*, 36(6):817–837, 2010.

[DJVV14] Joachim Denil, Maris Jukss, Clark Verbrugge, and Hans Vangheluwe. Search-Based Model Optimization Using Model Transformations. In *Proceedings of the 8th International Conference on System Analysis and Modeling (SAM'14)*, volume 8769 of *LNCS*, pages 80–95. Springer, 2014.

[EWZ14] Dionysios Efstathiou, James R. Williams, and Steffen Zschaler. Crepe complete: Multi-objective optimisation for your models. In *Proceedings of the 1st International Workshop on Combining Modelling with Search- and Example-Based Approaches (CMSEBA'14) @ MODELS*, volume 1340, pages 25–34. CEUR-WS.org, 2014.

[FTW15] Martin Fleck, Javier Troya, and Manuel Wimmer. Marrying Search-based Optimization and Model Transformation Technology. In *Proceedings of the 1st North American Search Based Software Engineering Symposium (NasBASE'15)*, 2015.

[MJ14] Hamid Masoud and Saeed Jalili. A clustering-based model for class responsibility assignment problem in object-oriented analysis. *Journal of Systems and Software*, 93(0):110 – 131, 2014.

[MVG06] Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

[YC79] Edward Yourdon and Larry L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice-Hall, Inc., 1st edition, 1979.