Method and Tool Support for Refinement of Test Suites

by

Nadia Roberta Traldi Spido

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Computer Science

at

Ottawa-Carleton Institute for Computer Science
School of Computer Science
CARLETON UNIVERSITY
Ottawa, ON, Canada K1S 5B6

January 2010

Copyright © 2010 by Nadia Roberta Traldi Spido

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-63828-6
Our file Notre référence
ISBN: 978-0-494-63828-6

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

■+■
Canada

ABSTRACT

When facing tight deadlines and rigorous quality assurance requirements, the manual refinement of test suites can offer testers a challenging assignment: analyzing and understanding large test suites, many times not even based on any specific rationale, and identifying their need to be simplified, transformed, or augmented. In order to help the tester to perform this task, Melba was introduced as a semi-automated method based on Machine Learning and Category-partition testing, aiming at helping the tester not only to understand test suites but also to improve them. However, previous studies have shown the need for stronger automation of the whole Melba process. In response to this need, we present here a study of the method along with improvements applied, in addition to the design and implementation of the Melba tool, developed in order to make test suites refinement with the Melba method an automated and straightforward activity.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my deepest gratitude to all who were part of this achievement, by helping me, supporting me, and offering me their incentive to do my best and complete this work, which was one of my dreams. I will never be grateful enough to God that gave me life and all the necessary conditions for me to achieve this goal.

To my dear supervisor Dr. Yvan Labiche and co-supervisor Dr. Lionel Briand, I would like to state there is no way to measure the richness of knowledge that they have gifted me with. I've learned precious lessons not only for research and academic life, but also lessons that I will take for life and that will help me to be a better professional wherever I am.

I have no words to describe how grateful I am for all the unconditional love and support that my husband Ronaldo has dedicated to me throughout this work as well as to all endeavours in our lives.

This work would never be complete without the most sincere words of thankfulness to my dear parents. They have raised me and given me education that made me capable of flying for my dreams. I will never forget that, for my inspiration comes from them.

My heartfelt appreciation goes also to Cesar for being more than my dear brother and offering me his friendship, support and love.

To all my family members not mentioned here, I am lucky to have them all, supporting and listening to me, and always offering me a friendly shoulder.

To my research partners and friends in Squall Lab, and to my dear friends and mentors in Canada and back in Brazil, I am grateful for their belief and offer of the most sincere motivation.

This is a unique moment in my life, and I am very glad I was always able to count on you all, and I wish we can always keep sharing valuable life experiences and growing together.

TABLE OF CONTENTS

1. INTRODUCTION	1
1.1. Problem Statement	2
1.2. Contributions	2
1.3. Thesis Contents	3
2. RELATED WORK	4
3. METHOD	7
3.1. Define Test Suite and Category-partition Specification	7
3.2. Generate Abstract Test Suite	8
3.2.1. Early Identification of Possible Problems	9
3.3. Generate Decision Tree	10
3.4. Analyzing DT Results	11
3.4.1. Observations	12
3.5. Refinements for Identified Problems	19
3.5.1. Missing Test Case	20
3.5.2. Redundant Test Case	21
4. AUTOMATION	22
4.1. Requirements	22
4.2. Functional Description	23
4.3. Conceptual Model	26
4.4. Design Decisions	30
4.4.1. Decisions Regarding the Test Suite Input	31
4.4.2. CP Specification and Abstract Test Suite	35
4.4.3. Data Model Persistence	35
4.4.4. Decision Tree	36
4.4.5. Analysis of DT	36
4.5. User Interface Design	36
4.5.1. Document Interface	37
4.5.2. Windows and Tabs	38
4.5.3. Menus and Menu Accelerators	40
4.5.4. Data Entry and Display Components	41
4.5.5. Control Components	43
4.6. Support Step by Step	45
4.6.1. Pre-defined Operations	45
4.6.2. Resources in Results GUI	46
5. CASE STUDY	49
5.1. System Specifications	50
5.1.1. The Taxi Billing System	50
5.2. Experiment 1	51
5.2.1. Experiment Design	52
5.2.2. Applying Melba method	54
5.2.3. Summarizing Resulting CP Specification	67
5.2.4. Summarizing Resulting Test Suite	67
5.2.5. Verification and Validation of the Melba Tool	71
5.3. Experiment 2	72
5.3.1. Experiment Design	72

5.3.2. Applying Melba method	76
5.3.3. Summarizing Resulting Test Suites	97
5.4. Experiment 3	102
5.4.1. Experiment Design.....	102
5.4.2. Applying Melba method	105
5.4.3. Summarizing Results	113
6. CONCLUSION	117
6.1. Future Work.....	119

LIST OF TABLES

Table 1 A small initial test suite for the Vending Machine system	8
Table 2 Abstract test suite for the Vending Machine TS proposed in Table 1	9
Table 3 Functional and non-functional requirements	23
Table 4 Mapping between requirements and use cases	26
Table 5 Applicable charges for the Taxi Billing System	51
Table 6 DT for Iteration 1 - Execution 2	66
Table 7 Comparison of performance of each TS for Experiment 1	68
Table 8 Requirements verified with the Experiment 1	72
Table 9 Comparison of performance of each TS for Experiment 2	74
Table 10 Comparison of performance of resulting TS and original ones for Experiment 2	97
Table 11 Combinations of choices and OEC exercised by more than 1 ATC in the same ATS	100
Table 12 Choices in C 2 combined with choices in the first missing combination	106
Table 13 Choices in C 4 combined with choices in the first missing combination	106
Table 14 Choices in C 2, other than Ch 2.1, combined with choices in C 1, C 4, C 5 ...	110
Table 15 Choices in C 3, other than Ch 3.1, combined with choices in C 1, C 4, C 5 ...	110
Table 16 Summary of Code Coverage and Mutation Scores for Experiment 3	113
Table 17 DTs Comparison	115

LIST OF FIGURES

Figure 1 Melba Process.....	7
Figure 2 Potential problems identifiable when generating ATS.....	9
Figure 3 Potential problems identified through the analysis of results.....	12
Figure 4 Adding Test Cases from the Tree.....	16
Figure 5 Example of the addition of test cases	17
Figure 6 Mapping of problems, refinements and improved artefacts.....	20
Figure 7 Use Case Diagram for Melba tool.....	24
Figure 8 Melba Input Conceptual Model.....	27
Figure 9 Melba Decision Tree and Analysis Conceptual Model.....	30
Figure 10 Melba tool - use of external resources.....	31
Figure 11 Choosing SUT to work with at the Melba tool.....	38
Figure 12 Specifying category partition (output equivalence class tab active)	39
Figure 13 Visualizing DT Results while modifying CP Specification	40
Figure 14 Main menus and options at Melba tool: File (a), View (b), Help (c).....	41
Figure 15 Labels and hints for text boxes in CP Spec definition. Partial window	43
Figure 16 Use of Hyperlinks as control components in results tabs. Partial window	45
Figure 17 Equivalence Class definition with pre-defined operation <i>exclusive interval</i> ...	46
Figure 18 Complete Melba GUI for DT Analysis	46
Figure 19 Parameters loaded from TS file into Melba Tool.....	54
Figure 20 Choice of Output Parameter for the ATS.....	55
Figure 21 Error message and list of test case values with multiple matches.....	55
Figure 22 ATS for output parameter totalCost (excerpt). TCs sorted alphabetically.....	57
Figure 23 DT and analysis of possible problems by Melba tool	58
Figure 24 Misclassifications for DT of Figure 23	59
Figure 25 Identification of OEC defined in CP Specification and not used in DT	61
Figure 26 Definition of new TC through Melba tool.....	63
Figure 27 Nonconformities identified for Iteration 3 of Execution 1	64
Figure 28 Nonconformities identified for Iteration 1 of Execution 2.....	65

Figure 29 Coverage of TSExec1_Init on the Taxi Billing code. Coverage details on the right, class TaxiDaoStructuralImpl.....	69
Figure 30 Summary of results for TS2_2	77
Figure 31 Summary of results for TS2_2 at Iteration 2	78
Figure 32 Correlation calculation for choices in categories C 3 and C 2, using TS2_2 ...	80
Figure 33 Correlation calculation for choices in categories C 4 and C 1, using TS2_2 ...	81
Figure 34 Summary of results for TS2_2 at Iteration 3	81
Figure 35 Summary of results for TS2_2 at Iteration 4	82
Figure 36 Misclassifications for TS2_2 at Iteration 4	83
Figure 37 Summary of results for TS2_3	83
Figure 38 Summary of results for TS2_3, Iteration 2	84
Figure 39 Correlation calculation for choices in categories C 2 and C 1, using TS2_3 ...	85
Figure 40 Correlation calculation for choices in categories C 4 and C 1, using TS2_3 ...	86
Figure 41 Summary of results for TS2_3, Iteration 3	87
Figure 42 Correlation calculation for choices in categories C 3 and C 2, using TS2_3 ...	87
Figure 43 Summary of results for TS2_3, Iteration 4	89
Figure 44 DT and Misclassifications at Iteration 4, with TS2_3	90
Figure 45 Correlation calculation for choices in categories C 4 and C 3, using TS2_3 ...	90
Figure 46 Summary of results for TS2_3, Iteration 5	92
Figure 47 Summary of results for TS2_3, Iteration 6	92
Figure 48 Correlation calculation for choices in categories C 3 and C 2, using TS2_3 ...	93
Figure 49 Correlation calculation for choices in categories C 5 and C 1, using TS2_3 ...	94
Figure 50 Summary of results for TS2_3, Iteration 7	96
Figure 51 Summary of results for TS2_3, Iteration 8	96
Figure 52 Misclassified ATCs for TS2_3 at Iteration 8	97
Figure 53 Sample DT: used category C 1 and unused category C 8	125
Figure 54 Observed Frequencies for categories C 8 and C 1	125
Figure 55 Expected frequencies (a) and test statistic values (b) for C 8 and C 1	126
Figure 56 Residuals (table), total test statistic and <i>P</i> value for categories C 8 and C 1 .	128
Figure 57 DT for Alternative Iteration 3 with category C 5 changed.....	144

LIST OF ACRONYMS

Acronym	Explanation
ATC	Abstract Test Case
ATS	Abstract Test Suite
CP Spec	Category-partition Specification
DT	Decision Tree
EC	Equivalence Class
GUI	Graphical User Interface
OEC	Output Equivalence Class
SUT	System Under Test
TC	Test Case
TS	Test Suite
UI	User Interface

1. INTRODUCTION

The process of refining a test suite, either by augmenting, modifying or reducing the set of Test Cases composing it, is similar to the process of re-engineering software, where instead of code it is from the test cases and system specifications that information is extracted, analyzed, and then used to decide about changes in the test suite. This process may be motivated by needs in several contexts of software development, such as in software evolution or in the reuse of open source software, where the software testing engineer (or tester) is faced with analyzing and understanding large test suites and trying to identify to what extent they exercise the target system functionalities and therefore whether they need to be simplified or augmented.

In order to assist the tester in performing this activity, Melba [5] was proposed as a method to be used in the context of black-box testing. Melba is based on Machine Learning (ML) and Category-partition (CP) testing, therefore the type of systems it proposes to deal with are those handled by CP, which is a black box testing specification method. Melba aims at helping the tester not only to understand a test suite that may need to be refined but also to improve it. When this task involves manual work on a large test suite, which may not be based on any specific rationale or specification, and especially when facing tight deadlines and numerous requirements of quality assurance, the tester responsible for this analysis may face a typical scenario of pressure, certainly subjected to mistakes. As a solution, having a method to follow specific steps can be of great help to the tester, as long as these steps are clear and efficient to apply. What is proposed by Melba is exactly solving this sort of problem by using a partially automated process.

The Melba method, as initially proposed [5], takes as inputs the test specification following the CP strategy and the test suite to be re-engineered. Based on the CP specification, the test suite is transformed into an abstract test suite composed of abstract test cases which are tuples of pairs (category, choice) associated with an output equivalence class (as opposed to concrete input/output values). The machine learning algorithm C4.5 is then used to learn from the abstract test suite input/output rules that relate pairs (category, choice), modeling input properties, to output equivalence classes. These rules form a decision tree that is finally analyzed to determine potential

improvements to the test suite (e.g., redundant test cases, need for additional test cases) as well as improvements to the CP specification (e.g., need to add a category or choice).

1.1. Problem Statement

When introducing and experimenting with the method [5], studies with large amounts of data have shown that Melba steps are too numerous and complex for the tester to apply manually, suggesting the need for a stronger automation. It is in this context that the Melba Tool started to emerge as a necessary instrument to make the Melba process applicable and helpful. Thus, in the current work we designed and developed this tool especially to make the re-engineering of test suites an automated and straightforward activity.

In addition to the lack of automation, our investigation showed that the method presented some deficiencies in terms of feedback on the relation between observations (nonconformities or indicators of potential problems, as detailed later in section 3.4.1) and the possible problems themselves, items that are all listed as our contributions in the next subsection.

1.2. Contributions

This thesis contributes to improvements of Melba in three different and complementary ways: contribution in terms of improvements to the Melba method; contribution in terms of tool support for Melba; contribution in terms of case study.

More specifically, improvements to the method pertain to:

- The use of category-partition constraints (properties and selectors) defined by the tester in the investigation of problems reported by Melba.
- The improvement of the analysis and feedback information (possible causes) linking the method observations and possible problems.
- The usage of the CP specification information in the analysis and suggestion of new test cases, as opposed to using exclusively the DT information as in the original method.

In terms of tool support, we first provide tool support for all the activities of Melba that were identified as being automatable in [5] by using available third party tools, frameworks and libraries. We also provide extensive tool support for the analysis of

Melba results. During the construction of tool support for Melba, we made a constant effort to create an easy to understand, learn and use graphical user interface, based on state of the art principles.

Last, we contribute to Melba by performing a case study composed of 3 different experiments where we evaluate different aspects of the method and validate the effectiveness of the tool.

1.3. Thesis Contents

The remaining of this Thesis is structured as follows: in Chapter 2 we investigate existent work related to the Melba method; Melba is detailed in Chapter 3, which also includes our contributions to Melba; Chapter 4 describes our tool supporting the method; our case study is discussed in Chapter 5; and conclusions are drawn in Chapter 6.

2. RELATED WORK

A number of approaches and tools relate to Melba but, as we shall see, none of them directly compares to Melba.

In [18] the authors propose a method aimed at avoiding the reduction of the fault detection effectiveness (FDE) of a test suite when reducing it, by using code coverage information of test cases to keep specific test cases in the reduced test suite even if they are redundant with respect to certain structural testing criteria. Their approach is named "Reduction with Selective Redundancy (RSR)" and cannot be directly compared to Melba since its main goal is to improve the FDE while reducing the TS (during regression testing), as opposed to Melba where we are interested in the functional, input-output relations exercised by a test suite. And fault detection and structural coverage are only two aspects we use to evaluate the TS quality. The main purpose of Melba is not test suite reduction, though it has this capability. Melba test suite reduction heuristic is simple compared to strategies devised to help regression testing, and is not currently supported by the tool, and future work will look at whether RSR can be integrated into Melba.

Daikon [11] is an implementation of dynamic invariant detection. A machine learning technique is applied to execution traces of a program and detecting these invariants helps in the understanding of the program, which can be helpful in test cases generation. Daikon mainly differentiates from Melba for dealing with the source code of the program, which is instrumented. Also, despite the fact that it permits an analysis of test cases, Daikon's output does not include any guidance or heuristics specific to the suggestion of test suite improvements.

Focused especially in the context of software evolution where regression testing is an essential activity, MATRIX [29] is a technique for augmenting a test suite. The strongest point of the method, as claimed by the authors, is its high effectiveness in identifying test cases with high capability of fault detection. MATRIX main purpose is to check whether an existing test suite adequately covers code changes and, if not, provide guidance for the creation of test cases that target those changes. This aspect is specific to regression testing, which makes this technique different from Melba that focuses on the system's specification definition, independently from any (recent) change that may have been applied to the code. In addition, a tool, MATRIXRELOADED, was implemented for the

application of MATRIX to programs written in Java. For being based on the source code of the system, it also distinguishes strongly from Melba which, based on a specification and a test suite, is independent from the programming language used in the target system.

In [20] the approach proposed is founded on Contract Driven Development and the main goal is to automatically describe test cases for a given software based on code contracts and executions of the software triggered by the user during its development (test-driven, agile development). While the user executes the unit under test, either directly or through a GUI, and without explicitly defining test cases (for instance JUnit tests), the method is to capture state traces and record those that lead to failures (as identified by broken contracts, the oracles). Each such execution, referred to as implicit test case by the authors, is then promoted to a concrete test case that can be re-run automatically. A tool called Cdd implements this method and targets Eiffel code since Eiffel's support for contracts is native and widely used. The deepest differences between this method and tool and Melba is the use of the code and its contracts and execution state traces, while Melba counts on a more high level (input-output relationship) specification.

In the specific context of TTCN-3 test suites, in [24] the authors propose an approach based on code smells to detect and improve test suite quality. The detection of code smells, inappropriate uses of code structures that could (and probably should) be better written, makes possible the assessment of the structural quality aspect of the test suite in terms of readability, maintainability, and reusability. Although this may also lead to improvements to the test cases, no such guidance is provided by the method. The tool implemented for this method, TRex (an Eclipse plug-in distributed under an Eclipse Public License), is obviously focused on syntactic problems, and a very valuable addition to Melba to check the quality of the input TTCN-3 file.

The method described in [21] is to derive unit tests from both a functional specification (under the form of contracts) and the code. First, the variables that influence the behaviour of the unit under test are identified from the specification: these are attributes, parameters and global variables. Then, depending on the types of the variables, the method is to create tests where variable values are selected to satisfy pre-defined relations: For instance, if two variables are integers, possible relations are $=$, $\neq$, $>$, $<$, $\leq$,

≥; Relations are also defined for other primitive types such as real, and data structures, such as array and object. Among a set of predefined relations, the specification and the code can be used to select a subset of relations that is appropriate to the context. In the proposed method, the selection and use (to identify test data) of these relations solely depends on the tester's decisions. Though most (if not all) these relations would likely be captured by applying Category-Partition, like in Melba, one important difference with Melba is that Melba supports the evolution and improvement of an existing test suite.

To sum up, Melba differs from the previous approaches in at least one of the following aspects: (I) it is independent from the software source code and therefore from the language in which it is implemented, (II) it works with a high level system specification, which also makes it more flexible, (III) it counts on a previously defined TS to be assessed, and (IV) it provides suggestions for quality improvement of the TS, be it by the addition, change or removal of test cases.

3. METHOD

Melba is an iterative process composed of five main steps as shown in Figure 1. The five subsections in this chapter bring the explanation of each one of those steps and when the process can be considered complete. Most of those aspects have already been defined in [5] and we will clearly indicate our own improvements/contributions to the method by underlying (parts of) sentences for simple contributions or indicating larger contributions in the margin of (parts of) paragraphs or sections.

Figure 1 Melba Process

3.1. Define Test Suite and Category-partition Specification

There are two pieces of information that the user is expected to give as input to the method: the test suite (TS) targeted for evaluation and possible improvements, and the specification of the System Under Test (SUT). Melba requires that each test case of the test suite be specified in terms of test input values and expected output values.

The Specifications of the system must follow the Category-partition (CP) method [27], including input and environment variables, their categories and choices¹. In addition, though this is not part of CP, the user must provide equivalence classes for output parameters. Though not part of the original Melba method, the user has to define constraints on choices, which is standard procedure when using CP. An example of a CP Specification with Categories, their choices, and the expected Output Equivalence Classes (OEC), is drawn in Appendix B.2 for the coin-based Vending Machine system described in Appendix B.1.

¹ Both parameters and environment variables will be referred to as parameters in the remainder of the text.

3.2. Generate Abstract Test Suite

Generating the abstract test suite (ATS) means creating a set of Abstract Test Cases (ATC) where each ATC corresponds to exactly one Test Case (TC) in the user's test suite. When creating an ATC for a TC, one looks at each parameter in sequence. For each parameter, and each of its categories (defined in the CP specification), one identifies which choice of that category corresponds to the value of the parameter for that test case. An abstract test case is therefore a series of pairs (category, choice), with as many pairs as categories in the CP specification, therefore identifying which conditions (in terms of choices' specification) applies to the test case. In addition one looks at each output parameter and identifies which output equivalence class corresponds to the test case. So an abstract test case is a series of pairs (category, choice) plus a pair (output parameter, equivalence class). Section 3.2.1 discusses situations where identifying whether a choice of a category (or an output equivalence class of an output parameter) is problematic.

To illustrate this, consider the coin-based Vending Machine system described in Appendix B.1, for which the CP Specification can be found in Appendix B.2. Assuming a TS as in Table 1, the resulting ATS is shown in Table 2. For instance, for parameter `value_inserted`, choices Ch 1.1, to Ch 1.7 correspond to values 0, 0.01, 0.05, 0.10, 0.25, 1 and 2 of the parameter, respectively. Therefore, test case TC1 involves choice Ch 1.5 and test case TC3 involves choice Ch 1.1, for example.

Vending Machine				
Test Cases	Input Parameters		Output Parameters	
	<code>value_inserted</code>	<code>item_wanted</code>	<code>product_released</code>	<code>refund_value</code>
TC1	0.25	"coca-cola"	""	0.25
TC2	1	"coca-cola"	"coca-cola"	0
TC3	0	"sprite"	""	0
TC4	2	"sprite"	"sprite"	1
TC5	0.01	"cookies"	""	0.01
TC6	0.05	"refund"	""	0.05

Table 1 A small initial test suite for the Vending Machine system

<i>Vending Machine</i>				
Abstract Test Cases	Categories		Output parameters	
	C1	C2	product_released	refund_value
	value_inserted	item_wanted		
ATC1	Ch 1.5	Ch 2.2	EC3	EC5
ATC2	Ch 1.6	Ch 2.2	EC1	EC4
ATC3	Ch 1.1	Ch 2.2	EC3	EC4
ATC4	Ch 1.7	Ch 2.2	EC1	EC6
ATC5	Ch 1.2	Ch 2.3	EC3	EC5
ATC6	Ch 1.3	Ch 2.5	EC3	EC5

Table 2 Abstract test suite for the Vending Machine TS proposed in Table 1

3.2.1. Early Identification of Possible Problems

Besides generating the ATS, this step also permits the early identification of some problems in the CP Specification as illustrated in Figure 2.

Figure 2 Potential problems identifiable when generating ATS

When building the ATS, more than one choice in the same Category may match the parameter value, which is a warning that the definitions of the choices in that Category are not exclusive, i.e. they overlap, contradicting a requirement of Category-partition definition [27]. A similar problem is also identifiable when trying to match the output parameter to an OEC: if an output parameter value matches more than one output equivalence class, then these classes are not disjoint and therefore they are not really equivalence classes. This situation could be caused by any one of the three problems shown in Figure 2, also described later in section 3.4. Ideally, the best moment to tell the user about this sort of problem would be while still providing CP definition, however due to the complexity that the choices and OECs can present, this would be very difficult to achieve.

Another possible analysis is to identify cases where an input parameter value does not match any of the choices in a specific category, or an output parameter value does not match any OEC for that parameter. In those cases a default choice (or OEC), N/A (Not

Applicable), is assigned. This means that no definition in the CP specification could be matched to that value. A N/A choice for a category is due to either (i) the defined choices failing to represent the whole domain of values of the category (problem II of Figure 2), which CP principles [27], or (ii) a category that cannot be combined with another category used in the (A)TC. A N/A output equivalence class is due to either (i) the OECs of the parameter failing to represent the whole domain of values of the parameter (problem II of Figure 2), which contradicts the notion of equivalence classes, or (ii) the output parameter not being an output of the (A)TC. This is further discussed in section 4.4.2.

3.3. Generate Decision Tree

Based on the ATS built with user's input information, a Machine Learning (ML) algorithm is triggered in order to draw a decision tree (DT). To run the machine learning algorithm one has to select one output parameter of interest so that the algorithm can identify input-output relationships between combinations of pairs (category, choice) and output equivalence classes for one parameter. It is important to select one parameter at a time since different input-output relationships, possibly with contradicting combinations of pairs (category, choice), may exist for different parameters. Selecting several output parameters would therefore not lead to useful, practical results.

The algorithm used is the C4.5 classification tree algorithm [28, 35]. It uses a ML technique that focuses on classification, which is the problem at hand in this method as we want to learn about the relationships between input parameters properties (Categories and Choices) and Output Equivalence Classes. In addition, it was chosen for its stepwise manner of partitioning the data set (in our context the set of Abstract Test Cases) using complex algorithms and heuristics to avoid overfitting the data with the goal of generating models that are as simple as possible. Other ML algorithms, like Ripper, are so-called covering algorithms that generate rules in a stepwise manner, removing observations that are "covered" by the rule at each step so that the next step works on a reduced set of observations. With coverage algorithms, rules are interdependent in the sense that they form a "decision list" where rules are supposed to be applicable in the order they were generated. Because this makes their interpretation more difficult, we will

use a classification tree algorithm, namely C4.5, and use the Weka tool [28, 35] to build and assess the trees.

Once the DT is generated, it is possible to analyze it by reading its paths (from the root node to any leaf). The nodes represent pairs (category, choice); a path contains a sequence of nodes and is associated to a leaf, i.e., an OEC. Each path can be interpreted as a specification of an input characteristics (combination of categories and choices) leading to an output equivalence class.

3.4. Analyzing DT Results

In order to point out potential problems in user's TS and/or CP Specification a set of verifications are performed. Figure 3 maps the identification of problems to possible observations from machine learning results. Grey elliptical elements represent eight problems that can be identified in the SUT data (either TS or CP) including two false positives (grey hatched elliptical elements). It is by observing Cases 1 to 5, the white circles, that it is possible to identify those problems, as detailed in Section 3.4.1. These observations are: (1) misclassifications, (2) a category, choice or OEC that is not used in the DT, (3) a combination of choices that does not appear in the DT, (4) a path in the DT has too many instances (i.e., abstract test cases), and (5) a path in the tree does not have any instance. Section 3.5 presents the possible refinements to CP and/or TS to address those problems (except the false positives). The false positives are so named because they are problems suggested by some heuristics that can be discarded by further analyses. In fact, these are situations that are identified as problems by the original Melba method but that we propose to discard. We decided to keep that information available to the user to provide as much information as possible, as described in more details in Section 3.4.1.3.

During the iteration process of Figure 1, iterations are performed until no instance of Cases 1 to 5 can be identified. The resulting TS and CP are then considered satisfactory. However, it may not always be possible to obtain exactly zero instance of Cases 1 to 5 because of the false positives. Therefore we suggest the process of Figure 1 continues until only false positives remain, that is until only instances of cases leading to false positives occur (i.e., Case 3 Missing Combinations of Choices or Case 5 Empty Leaves).

Figure 3 Potential problems identified through the analysis of results

3.4.1. Observations

Cases 1 to 5 in Figure 3 are discussed in sections 3.4.1.1 to 3.4.1.5.

3.4.1.1. Case 1: Misclassifications

A misclassification is an instance of an abstract test case that is classified in a path of the DT, i.e., with a specific OEC, but this OEC is different from the expected one for that abstract test case.

The number of misclassifications can be identified from the DT or from the confusion matrix, and Weka (the ML algorithm) reports on the misclassified instances (ATCs). As pointed out in [5], this may be due to **Problem I** Missing Category or Choice and **Problem II** Ill-defined Choice or OEC. Unfortunately, it does not seem to be possible, at the present time, to automatically identify which is the real problem. However, a heuristic to decide which one is the likely problem is to identify how many times a choice (or OEC) is involved in misclassified ATCs. The rationale is that the larger the number of misclassified ATCs involving a choice (or OEC), the more likely the choice (or OEC) is the reason (ill-defined choice/OEC) of the misclassifications.

3.4.1.2.Case 2: Unused Categories, Choices, or Output Equivalence Classes

There are three possible causes for Case 2: **Problem II** Ill-defined Choice or OEC, **Problem III** Useless Category, Choice, or OEC, and **Problem IV** Missing Test Case. In specific situations, it is possible to automatically eliminate **Problem IV** as a possible cause but, as explained next, in most of the cases it is not likely to automatically eliminate any of them, or to suggest only one of them as the exact cause.

In order to identify unused CP elements, i.e. choices (or even all choices in a Category) and OECs, both ATS and DT analyses are necessary. First, if an element is not used in at least one abstract test case, then it is pointed out as an occurrence of Case 2. The cause for this occurrence is likely to be **Problem IV** Missing Test Case. However, the CP Specification could be not properly defined and this unused element could be incorrect. As a result, all three possible causes mentioned previously for this case should be investigated, with the additional information that the element was not used in the ATS.

In case the element is used in at least one ATC, this is an occurrence of Case 2 with the additional information that the element is used in the one ATC but not in the DT, this time excluding **Problem IV** Missing Test Case as a possible cause.

A category or choice may not appear in the DT (though it is used in the ATS) if that category (or choice) does not characterize inputs that can change the value of the selected output parameter. That category may instead characterize inputs that can change the value of another parameter. An instance of this situation will be discussed in our case study (very end of Section 5.2.2.1).

Finally, if the unused element is a choice, it may simply not appear in the DT because each time it appears in ATCs, another choice (from another category) also appears systematically and the ML algorithm used this other choice instead in the DT. One could say that the two categories are (strongly) correlated. In our context where variables being compared (i.e., categories) are qualitative variables (i.e., categories have discrete values, the choices), the statistics terminology is that there is an association between qualitative

variables² (categories). When building the DT, at each node of the tree the ML algorithm chooses the (category, choice) pair (or the attribute, if we use the ML terminology) that best splits the set of ATCs (or instances) into subsets of an OEC (or simply a class). The chosen (category, choice) pair is the one with the highest normalized information gain. Information gain measures how well a given attribute separates the instances according to their target classification, i.e., it measures the expected reduction in entropy, where the entropy measures the impurity in a collection of instances. Assume the ML algorithm has the choice between two attributes (i.e., two (category, choice) pairs) such that if the first attribute is selected for splitting the instances, this results in instances sets (one for each attribute value) that each has exactly one target classification (i.e., one value of the OEC), and if the second attribute is selected for splitting the instances, this results in instances sets that each has several target classifications. Then, the first attribute results in purer instances sets than the latter (smaller diversity in classifications), i.e., it reduces most the entropy (there is more information gain). The first attribute will be selected for splitting.

In the case there is an association between two (category, choice) pairs, the information gain for the two pairs is the same. If the algorithm selects one pair to split instances, when trying to split in later phases, the second pair will not bring any information gain and therefore will not be selected. Therefore, when we have a (category, choice) pair that is not used in the DT, our approach is to verify the level of association of this pair with all other (category, choice) pairs present in the DT. In this way, we can add some information that could explain why that (category, choice) pair, or the whole category, was not used in the DT and the user can evaluate applicable changes to the TS that would eliminate that problem.

The verification of association between each two candidate categories means verifying whether there is any relationship between them according to the test of association for qualitative variables [3]. When we compute the association between two qualitative variables, or two categories³ A and B where B is used in the DT but A is not,

² The statistics terminology in the literature also refers to those as categorical variables, but as we already have the notion of category in this thesis (with a different meaning), we opted for the name qualitative variables.

³ From this point we will simply refer to qualitative variables as categories.

we need to follow the steps of an appropriate test of association according to the sample size, i.e. the number of instances, or ATCs in the ATS. The method used here is the χ^2 (chi-squared) distribution [3], with the identification of residuals and application of necessary adjustments. This method allows one to identify whether the proportions of instances that involve each choice of category A differ according to the choices of category B. If they don't, we can say that the two categories have an association (one would say they are correlated). The use of the χ^2 (chi-squared) distribution is discussed in details, with a running example, in APPENDIX A.

3.4.1.3.Case 3: Missing Combinations of Choices

A Missing Combination of Choices, reported by the Melba process, consists of a set of x choices ($x \leq n$ where n is the number of categories defined in the CP specification), representing part of a candidate ATC which is considered to be missing in the ATS. Different approaches can be considered for suggesting missing combinations of choices to the user, and we discuss two of them below.

The most obvious way of identifying missing choice combinations is perhaps to start from CP Specification and identify all possible combinations of choices considering applicable properties and selectors, and for each of those combinations, verify whether there is at least one ATC in the ATS that exercises this combination. If no ATC exercises a combination, this suggests that a Test Case for that combination is possibly missing or that the CP Specification has problems (e.g., it may have useless choices or it may miss properties or selectors). One advantage of this procedure is that one can exhaustively investigate all the feasible combinations of choices. One drawback is that, though there are techniques and algorithms to automate that process, obtaining a minimum number of choice combinations (and therefore a minimum number of test cases, which is a good idea cost-wise) is still an open problem in the research community. Another issue is that this strategy does not benefit from what can be observed in the DT that is what the ML algorithm has learnt.

Another approach, which is part of the Melba method [5], is to explicitly rely on the DT. It consists in identifying, from the DT, which combinations of choices may be relevant to determine the output class and could be missing from the test suite. Assume that part of the tree obtained from the ML algorithm shows categories C_1 , C_2 , and C_3

with choices Ch 1.1 and Ch 1.2, Ch 2.1 and Ch 2.2, and Ch 3.1 and Ch 3.2, respectively, as illustrated in Figure 4 (a). The tree excerpt indicates that combining Ch 1.2 of category C 1 with Ch 3.1 or Ch 3.2 of category C 3 plays a role in determining output equivalence classes (the pairs of choices belong to different paths in the tree). Since C 1 has another choice than Ch 1.2, namely Ch 1.1, we may conjecture that C 3 might also be relevant to determine the output in the context of Ch 1.1 and that the tester should therefore try to combine choice Ch 1.1 with C 3's choices. Similarly, the tree suggests that the combinations of Ch 1.2 with C 2's choices may be missing in the test suite, thus resulting in four test cases being added⁴. This heuristic can be generalized to cases where category C 1 is not a *parent* of C 2 and C 3 in the tree but rather an *ancestor* of C 2 and C 3 (i.e., there are intermediate categories): Figure 4 (b).

One advantage of this heuristic is that by using the information provided by the tree, when intending to cover new choices, the tester does not have only the expensive option to exercise all the feasible combinations of choices, but can focus on combinations that are likely to affect the output.

Figure 4 Adding Test Cases from the Tree

To illustrate this heuristic on an example, consider the decision tree of Figure 5 (a). According to the previous heuristic, it suggests that choices Ch 2.2, Ch 2.8 and Ch 2.3 from category C 2 should be combined with choices of category C 4 that are under the branch of Ch 2.4 in the tree. Similarly, choices Ch 2.8, Ch 2.3 and Ch 2.4 from category C 2 should be combined with choices of category C 3 that are under the branch of Ch 2.2

⁴ There is one exception though: if Ch 1.1 is an error condition (e.g., an out of range input value), then Ch 1.1 is not combined with Ch 3.1 and Ch 3.2. This is consistent with the CP strategy.

in the tree. This results in a number of missing combinations of choices: Figure 5 (b) lists these suggested combinations.

C 5 = Ch 5.2	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.2) (C 4,Ch 4.2)]
C 1 = Ch 1.1	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.2) (C 4,Ch 4.1)]
C 2 = Ch 2.2	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.8) (C 3,Ch 3.2)]
C 3 = Ch 3.2: EC 6 (2.0)	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.8) (C 3,Ch 3.1)]
C 3 = Ch 3.1: EC 4 (1.0)	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.8) (C 3,Ch 3.3)]
C 3 = Ch 3.3: EC 7 (2.0)	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.8) (C 4,Ch 4.2)]
C 2 = Ch 2.8: EC 6 (2.0)	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.8) (C 4,Ch 4.1)]
C 2 = Ch 2.3: EC 6 (3.0)	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.3) (C 3,Ch 3.2)]
C 2 = Ch 2.4	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.3) (C 3,Ch 3.1)]
C 4 = Ch 4.2: EC 5 (2.0)	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.3) (C 3,Ch 3.3)]
C 4 = Ch 4.1: EC 6 (2.0)	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.3) (C 4,Ch 4.2)]
	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.3) (C 4,Ch 4.1)]
	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.4) (C 3,Ch 3.2)]
	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.4) (C 3,Ch 3.1)]
	[(C 5,Ch 5.2) (C 1,Ch 1.1) (C 2,Ch 2.4) (C 3,Ch 3.3)]
(a)	(b)

Figure 5 Example of the addition of test cases

Several verifications can be performed when those possibly missing combinations of choices are identified. First, one can verify whether the combination, in addition to being absent from the DT, is absent from the ATS. In case it is already used in the ATS, this is not really a missing combination: this is an occurrence of Case 3 (Missing combination of choices) that results in a false positive, specifically False Positive FP 2 (Combination already used in Test Cases).

If instead, the combination is absent from the ATS, one can then check whether it is feasible based on the constraints of the CP specification: If the combination involves a Single or an Error Choice and that choice is already involved in an ATC, then this combination is not feasible (Single and Error choices are involved in only one test case, and therefore ATC in the test suite); If the combination does not satisfy the constraints (properties and selectors) of the choices it involves, then it is unfeasible. In these two cases, the combination is an instance of Case 3 that results in a false positive, specifically **False Positive FP 1 (Unfeasible Combination of Choices)**; Otherwise, the combination is

indeed a feasible missing combination of choices. In this case, the combination is an occurrence of Case 3 resulting from **Problem II** (Ill-defined Choice or OEC), **Problem IV** (Missing Test Case), or **Problem V** (Ill-defined Properties and/or Selectors).

We believe that it is important to record and indicate those false positives to the user, instead of simply detecting them and discarding them (i.e., not showing them to the user) for two main reasons: (1) Those false positives result from analyses of the DT, which is an important piece of information to show to the user. If we detect those false positives and discard them (without showing them to the user), the user may identify them (by other means) and may then be surprised that they seem to not detected; (2) This information on false positives may still be relevant to the user who may re-think the CP specification (categories, choices, properties and selectors). For example, when provided with an unfeasible combination of choices (unfeasible according to properties and selectors), the user may realize that this combination does make sense and is feasible (considering the specification of the system), thereby suggesting problems in the definition of properties and selectors.

3.4.1.4. Case 4: Too Many Test Cases for a Rule

By looking at the number of instances, or test cases, correctly classified in a certain leaf of the DT, it is possible to identify occurrences of rules, or combinations of choices, that may be exercised more than necessary, i.e. in case that number is greater than 1 it may suggest the existence of redundant test cases since they all match the same path in the tree, i.e., the same input/output characteristics. As shown in Figure 3, **Problem I** (Missing Category or Choice), **Problem II** (Ill-defined Choice or OEC), and **Problem VI** (Redundant Test Case) are the possible causes for this case.

3.4.1.5. Case 5: Empty Leaves

An Empty Leaf is a DT leaf where no instance, or Test Case, was classified. This suggests that a (category, choice) pair was found to be significant by the decision tree algorithm to split the set of ATCs, and that the resulting combination of choices (i.e., path in the DT) is not exercised by the ATS. Similarly to a Missing Combination of Choices, an empty leaf (i.e., the path in the DT) indicates a combination of x choices ($x \leq n$, n being

the number of categories), representing (part of) an ATC which seems to be missing in the ATS. The same investigations as in section 3.4.1.3 can be conducted.

Note though that there is a difference between Case 5 and Case 3 (section 3.4.1.3): An empty leaf potentially indicates a missing combination but also a matching output equivalence class (OEC), which is not the case of Case 3. Similarly to Case 3, the missing combination suggested by the empty leaf may not be feasible (because of constraints between choices). One difference with Case 3 is that even if the suggested missing combination is feasible, finding a test case matching the empty leaf may not be feasible because a test case matching the missing combination may not have the expected OEC (i.e., the OEC indicated in the leaf).

3.5. Refinements for Identified Problems

Refinements are suggested according to the types of problems identified through the observations. Figure 6 illustrates a refinement for each problem, and the artefact (CP or TS) being refined.

The user needs to manually analyze the CP specification when the Melba method suggests it needs refinements: in the case of a Missing Category or Choice, Ill-defined Choice or OEC, Useless Category, Choice, or OEC, Ill-defined Properties and/or Selectors. Refinements can be the addition of a category, the addition of a choice, the modification of a choice (or choices) or OEC, the deletion of a category, choice or OEC, the modification of properties or selectors.

Refinements to TS, i.e., missing test case and redundant test case, need further discussion since different alternative heuristics can be investigated: sections 3.5.1 and 3.5.2, respectively.

Figure 6 Mapping of problems, refinements and improved artefacts

3.5.1. Missing Test Case

A missing test case is specified as a missing combination of x choices, $x \leq n$ where n is the number of categories in the CP specification. When $x < n$, the (missing) combination of choices is not complete to allow its instantiation into a test case: choices for $n-x$ categories need to be identified. Different heuristics can be considered and investigated for selecting the missing choices, and we mention three of them here. Without loss of generality, let's assume that the missing (incomplete) combination of choices is $Ch_1, Ch_2, \dots, Ch_x$, each choice Ch_i belonging to a different category C_i , $i=1, \dots, x$, $x < n$, respectively. Therefore, one needs to identify one choice for each of the categories $C_{x+1}, \dots, C_n$.

Heuristic 1: Inspired from criterion Base-choice coverage. Base-choice coverage is one of the possible criteria to use during Category-Partition testing, assuming a base (or nominal) choice is chosen for each category. Using this criterion, a base test is formed by using the base choice for each category, and subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other category. Following this principle, one can select the base choice for each of the missing category $C_{x+1}, \dots, C_n$ in the missing, incomplete combination of choices. The rationale is to keep as much of the newly created (abstract) test case nominal as possible.

Heuristic 2: Selecting choices that are not already combined in the ATS with the choices of the missing combination. One can count the number of times, in the initial ATS, each choice of each missing category $C_{x+1}, \dots, C_n$ is combined with choices from the missing combination, i.e., $Ch_1, Ch_2, \dots, Ch_x$. For instance, assuming category C_{x+1} has m choices, we obtain m counts $c_1, \dots, c_m$ where c_i is the number of times choice i of C_{x+1} appears in combination with $Ch_1, Ch_2, \dots$ or Ch_x in ATS. Heuristic 2 is then to select a choice for each $C_{x+1}, \dots, C_n$ that has the smallest count value. The rationale is to obtain an abstract test case that exercises as many new combinations of choices as possible.

Heuristic 3: Selecting choices that are already combined often in the ATS with the choices of the missing combination. Reusing the notion of counts discussed above, heuristic 3 is to select a choice for each $C_{x+1}, \dots, C_n$ that has the largest count value. The rationale is to obtain an abstract test case that exercises as few new combinations of choices as possible.

Note that each of these three heuristics may lead to a combination of n category choices that is unfeasible given the choices constraints. If such a situation, one can select one fewer base choice in the case of heuristic 1, the second smallest count value in the case of heuristic 2, the second largest value in the case of heuristic 3.

3.5.2. Redundant Test Case

Handling redundant test cases means first identifying a threshold number above which the number of instances in a leaf is assumed to indicate redundancy. In [5] that number was 1: any leaf in the DT with a number of instances strictly above 1 was considered to indicate redundant test cases. Once a leaf is identified as indicating redundancy, a heuristic can be used to reduce that redundancy, i.e., to select ATCs (i.e., TCs) to be removed from the ATS (TS). Several heuristics can be considered. The one used in [5] was to randomly remove a redundant ATC (TC), one at a time, until the structural coverage of the reduced test suite, measured for instance in terms of edge and statement coverage, drops. This heuristic has shown not be sufficiently efficient [5] and therefore removing test cases needs further investigation of additional heuristics, which will not be investigated in this thesis.

4. AUTOMATION

This section describes the tool that was designed and implemented to automate the Melba process. The first subsection starts by stating the requirements for this automation. Section 4.2 brings Melba use cases diagram and descriptions. Section 4.3 presents the conceptual model of the system. Section 4.4 describes our design decisions, introducing all libraries and external tools used. Section 4.5 is about the graphical user interface design decisions made. Section 4.6 describes how the tool is intended to support step by step the decisions the user needs to make in the Melba process.

4.1. Requirements

The objectives of providing an automated tool for the process were motivated by difficulties in the previous attempt [5] to apply the method to large systems, even using a partially automated solution. Based on that, the main goal was to make the method easier to apply by automating as many activities as possible, which can be decomposed into the following functional and non-functional requirements (FR and NFR, respectively): Table 3.

	Acronym	Description
Functional Requirements	FR1	To load a pre-defined test suite
	FR2	To load CP Specification from user's definitions
	FR3	To generate fully automatically the abstract test suite
	FR4	To generate automatically the Decision Tree corresponding to the input data
	FR5	To support user's decision-making process by reporting every identified results of possible problems, and facilitating the recognition of possible causes
	FR6	To provide direct application of suggested improvements to both CP Specification and TS
	FR7	To save, export, keep tracking and recover system data along the fine tuning process
Non-Functional Requirements	NFR1	To use an industry-standardized format for the test suite to be loaded
	NFR2	To provide a flexible way for the user to define expressions for choices and output equivalence classes and to have them syntactically validated
	NFR3	To hide complexity of the interface with the DT generator algorithm in Weka, by executing and interacting automatically with it
	NFR4	The Graphical User Interface should be designed by following state of the art principles in order to be easy to understand, learn and use

Table 3 Functional and non-functional requirements

4.2. Functional Description

The target user of Melba is a human actor interested in evaluating the quality of, and possibly improving, a given system test suite; typically a test engineer. This user is responsible for defining not only the specification for the system whose test suite is to be evaluated (using CP), but also the initial test suite. In order to be able to define these input data, and according to the current decisions in our method (Section 4.4), it is hence assumed that the user has knowledge on both CP and the format of the input file specifying the test suite (TTCN-3, as discussed in 4.4).

The Use Case diagram in Figure 7 shows Melba use cases. UC1, UC2, UC3, UC4, and UC5 are at the highest level and represent the user's main goals in the process of using Melba. The first one (UC1) is about loading the test suite in a TTCN-3 format and semi-automatically specifying the details of the CP Specification (input and output parameters, categories and choices) and output equivalence classes. UC2 is about the core of the Melba process, from creating the ATS, invoking the Machine Learning algorithm,

and analyzing results. The use cases UC3 and UC4 allow the tester to modify the CP Specification and/or the TS. And the last high-level use case (UC5) is about saving the data.

Figure 7 Use Case Diagram for Melba tool

UC1 is a generalization of three use cases: loading the test suite (UC1.1), defining the input parameters and environment variables specifications (UC1.2), and defining the output parameters specifications (UC1.3). In UC1.1, the user provides the test suite (TS) that should be evaluated (see the discussion in Section 4.4.1 about its format), and the tool can automatically identify parameters and their types from the TS file and load them into the CP Specification, or optionally the user can first define these parameters directly into the tool (UC1.4) and load the test suite file later. In the latter case, when performing UC1.1, the compatibility of these parameters (types and names) with the ones from the TS is verified (UC1.5). The user is then alerted of any incompatibility of parameters and can choose to accepted the file and the necessary modifications: in case there are extra parameters in the CP Specification that are not in the TS file, these parameters are removed from the CP Specification as well as any category, choices or output

equivalence classes linked to them; in case the TS file contains extra parameters, they are imported and added to the CP Specification. In UC1.2, the tester has to define categories and choices for each input parameter (UC1.2.1) by using either UC1.6, the operations pre-defined according to the parameter data type (see Section 4.6.1), or UC1.7, by writing Java expressions that are compiled at run time (see Section 4.4.2). As the definition of the categories and choices depends exclusively on the parameters definition, it can be done either before or after loading the TS file. When defining the choices, one of the choices must be selected as a base choice [1] (i.e., a nominal choice), UC1.2.2, which may be useful if a base choice coverage strategy is used in the context of CP-based testing. Similarly to UC1.2, in UC1.3 the tester is supposed to define the output equivalence classes, UC1.3.1 (using either operations, UC1.6, or Java expressions, UC1.7) for output parameters, which, similarly to parameters, are either defined in the tool or imported from the TS file.

UC2 includes the actions that can be taken once the input is considered ready and the user is prepared to analyze possible problems and suggested changes. This part is entirely automated. It starts by generating the abstract test suite (UC2.1), then using the decision tree algorithm (UC2.2), and finally analyzing obtained results (UC2.3). Potential problems can be found at any of those steps and once identified they are presented to the user. UC2.1.1 verifies problems related to the CP Specification as a result of UC2.1 (Section 3.2.1), when the user must correct any problem before proceeding with the analysis. In the same way for UC2.2, in case problems are found in the data that are going to be used by the ML algorithm (e.g., item 10 in Section 4.4.1.2), UC2.2.1 captures and reports these problems to the user before any analysis results can be reached. It is different in the case of UC2.3 where any potential problem identified at UC2.3.1 is automatically analyzed for possible root causes (UC2.3.2) and presented to the user together with some typical suggestions of changes, if any, that would potentially fix the problem (UC2.3.3). The user is then supposed to analyze each case and decide for solutions.

The use cases UC3 and UC4 are providing support for the user to modify the CP specification and/or the test suite, with different kinds of tool support, as described later in Section 4.6. In UC3, proposed changes to the CP Specification are related to the

addition, deletion, or modification of categories, choices, and OECs, all performed through use cases UC1.2.1 and UC1.3.1, while the changes in UC4 are the definitions and additions of possibly missing test cases, UC4.1. Note that the tool does not currently support the deletion of test cases when some are deemed redundant as removing test cases needs further investigation of additional and efficient heuristics, which is out of the scope of this thesis. And the remaining use case, UC5, is the option for persistence of data which is available to the user and provides version control and recovery resources (see Section 4.4.3).

The use cases presented above provide a complete environment for test engineers to refine a given software specification and test suite using the Melba method.

These use cases implement one or more of the requirements, as illustrated by Table 4.

	Acronym	Use Case implementing requirement
Functional Requirements	<i>FR1</i>	UC1.1, UC1.5
	<i>FR2</i>	UC1.2, UC1.2.1, UC1.2.2, , UC1.3, UC1.3.1, UC1.4, UC1.6 and UC1.7
	<i>FR3</i>	UC2.1 and UC2.1.1
	<i>FR4</i>	UC2.2 and UC2.2.1
	<i>FR5</i>	UC2.3, UC2.3.1, UC2.3.2 and UC2.3.3
	<i>FR6</i>	UC3 (through UC1.2.1 and UC1.3.1), UC4 (through UC4.1)
	<i>FR7</i>	UC5
Non-Functional Requirements	<i>NFR1</i>	UC1.1
	<i>NFR2</i>	UC1.6 and UC1.7
	<i>NFR3</i>	UC2.2 and UC2.2.1
	<i>NFR4</i>	All UCs triggered by the actor (GUI in general, see Section 4.5)

Table 4 Mapping between requirements and use cases

4.3. Conceptual Model

This Section describes the conceptual model of the Melba tool, identifying all manipulated concepts and their relationships. The motivation here is to describe how all the concepts relate to each other in a form that is rigorous, precise, and reusable during further development steps of the Melba tool. Two class diagrams are used to illustrate that (Figure 8 and Figure 9), for the sake of clarifying the presentation, though they correspond to one class diagram, as visible from the common classes in the two diagrams. Also, not all the attributes and operations are shown in the diagrams.

Figure 8 Melba Input Conceptual Model

In Figure 8, we model the inputs of the tool together with the resulting abstract test suite. Some inputs are imported from the TTCN-3 test suite (e.g. TestCase) whereas others are provided by the tester (e.g. Choice), as described in the use case model above. This is denoted with the stereotypes «TTCN-3» and «Tester», respectively. Parameter can be either imported from the TS file, or provided by the user and later matched to the loaded TS file. We decided to denote it with the «TTCN-3» stereotype. This class diagram can be briefly explained as follows: the SystemUnderTest is associated with a TestSuite, a CP Specification (CPSpec), and eventually an abstract test suite (AbstractTS). A TestSuite is composed of Records, Parameters, and their Types, following the TTCN-3 standard and Melba-specific restrictions detailed in section 4.4.1. Each Type has a set of supported pre-defined Operations (section 4.6.1) which the tester can use to define the expressions for Choices and OutputEquivalenceClasses (associations between Choice and Expression and

between `OutputEquivalenceClass` and `Expression`). Alternatively, the user can define `Choices` and `OutputEquivalenceClasses` using Java expressions (class `JavaExpression`). In addition a `TestSuite` is composed of `TestCases`, which provide parameter values (`ParamValue`). The CP Specification is composed of `Parameters` and their `Classifications` (`Category` and `OutputEquivalenceClass`), besides optional properties (`Property`) and selectors (`Selector`) applicable to `Choices`. A `Parameter` instance represents an input parameter of the SUT (a parameter in its signature), an environment variable, or an output parameter of the SUT, which is specified by attribute `type` of type `ParamType` (enumeration). A `Category` is associated to at least one `Choice`. As defined in Section 3.2, an abstract test suite (`AbstractTS`) is composed of abstract test cases (`AbstractTC`), which are involved in a one-to-one relationship with `TestCase`. An `AbstractTC` is a tuple of choices for input parameters and one equivalence class for the output parameter selected. (Though one output equivalence class can be identified for each parameter for each abstract test, as discussed in section 3.2, the machine learning algorithm can only identify input-output relationships between input parameters and one output parameter, as mentioned in section 3.3.) The `AbstractTS` can also be associated to `Correlation` information, which is created as described in section 3.4.1.2. Each instance of `Correlation` is associated to a set of abstract test cases (recall section 3.4.1.2) and contains the results for the test of association between two categories, and each of its `CorrelationElements` contain the specific results associated to each pair of choices in those categories.

Figure 9 shows that a `SystemUnderTest` is also associated with decision tree analyses⁵ (`DecisionTreeAnalysis`). An analysis results in a decision tree, which is modeled by classes `DecisionTreeNode` and `DecisionTreeLeaf` and their relationships: a `DecisionTreeAnalysis` is linked to the root node (role name `rootNode`) of the corresponding decision tree. A `DecisionTreeLeaf`, more specifically the path from that leaf to the root of the tree, classifies a number (zero or

⁵ Although the multiplicity in the diagram shows that more than one analysis can be performed, only one (the most recent one) is currently recorded (the implemented multiplicity is 1).

more) of `AbstractTCs` (association between `DecisionTreeLeaf` and `AbstractTC`), and may show misclassified instances in the tree (attribute `incorrectlyClassified`). An analysis may report nonconformities (`Nonconformity`), i.e., signs of potential problems, which can be of different types as previously explained (section 3.4): a `Misclassification` is a misclassified `AbstractTC` and relates to a set of decision tree nodes (role name `treePath`) indicating where in the DT that `AbstractTC` was misclassified; an `UnusedElement` can be an unused category or choice (`UnusedCategoryChoice`), which is one⁶ choice from the CP specification, or an unused output equivalent class (`UnusedEqClass`), that is an Output Equivalence Class from the CP specification; a `MissingChoicesCombination` is associated to a set of choices composing the combination (i.e., a path in the DT); and an `EmptyBranch`, which is a special case of `MissingChoicesCombination`, being also associated with an Output Equivalence Class (the empty leaf in that path in the DT). Each of those nonconformities is linked to a set of `PossibleCauses`, which are of different types (enumeration `CauseType`) specifically, problems and false positives (first introduced in Section 3.4).

⁶ An unused category results in all its choices being unused. In the case of an unused category we record all its unused choices as separate instances of `UnusedCategoryChoice`.

Figure 9 Melba Decision Tree and Analysis Conceptual Model

Note that the discussion of Figure 8 and Figure 9 above suggests that there are constraints between instances of the classes: for instance, there are as many abstract test cases as test cases for a SUT. These constraints are summarized in APPENDIX C, where constraints are further specified with the Object Constraint Language.

4.4. Design Decisions

The programming language chosen for implementing the tool was Java with the support of Swing/AWT resources in order to produce a suitable GUI. Moreover, the use of other tools, frameworks, and libraries was necessary to facilitate the completion of some specific tasks. Figure 10 gives an overview of the architecture of the Melba tool, showing its dependencies to those external resources. Figure 10 also shows (in grey boxes) the purpose of these external resources. The next subsections discuss those resources.

Figure 10 Melba tool - use of external resources

4.4.1. Decisions Regarding the Test Suite Input

In this section we discuss the choice of a format to help the user provide test cases (section 4.4.1.1), the Melba-specific restrictions that apply to this format (section 4.4.1.2), and the technology we use to parse test case descriptions under this format (section 4.4.1.3).

Recall from the description of Melba that Melba needs to know input and output parameters names and types, and for each test case the input and output parameter values.

4.4.1.1. Test Suite Input Format

In order not to define a new format for the TS that would be exclusive to Melba, we investigated two alternative standard formats. Besides providing an already defined format (that we do not have to define ourselves), they offer the possibility to use test suites that have been originally defined for other purposes (i.e., the input TS to Melba is not created specifically for Melba). The options we analyzed were TTCN-3 (Testing and Test Control Notation Version 3) [34], and UTP (UML Testing Profile) [26]. TTCN-3 is an internationally standardized testing language that has been used for more than 15 years in industry. It applies to a variety of application domains and types of testing. UTP

defines a test modeling language, based on UML, for designing, visualizing, specifying, analyzing, constructing, and documenting the artefacts of test systems. It is also applicable to testing systems in various application domains. According to our analysis, TTCN-3 was chosen not only because it seems more diffused than UTP in the industry, but also because it leads to test suite of less (structural) complexity since it does not involve models.

Note that the Melba method is independent from this choice of input file format, and other general, domain specific or organization specific format can be easily considered. Indeed, the only information Melba needs is (1) the types of input and output parameters, and (2) the input values and expected output values for those parameters that are used in test cases. Therefore, other formats could be investigated, such as JUnit, as long as the required information can be extracted from the test suite.

4.4.1.2. Melba Restrictions for the TTCN-3 TS File

TTCN-3 syntax and operational semantics are very wide and definitions can become especially complex to parse. For that reason, it was decided to apply some restrictions on the use of TTCN-3 language resources in the first version of the Melba tool, to which the TS file must comply in order to be accepted and loaded into the tool. APPENDIX D shows a sample TTCN-3 (TS) file suitable to the Melba tool. Before we list the restrictions applied, we first summarize the main concepts of TTCN-3.

The main building block of a TTCN-3 specification is a *module*. A test suite is a module, and modules can be understood as units of compilation with a number of characteristics: they can be parameterized (receive external values); they can import definitions from other modules (reuse structures defined in other modules); they can have attributes (such as encoding directives, display information, or user-specific extensions); and they contain a Definitions part and an optional Control part as explained next.

Module Definitions: This part contains definitions that are global to the entire module:

- Type Definitions, based on the TTCN-3 predefined primitive types: The primitive types comprehend basic data types (such as *integer*, *float*, *boolean*), basic string types (*bitstring*, *hexstring*, *octetstring*), character string types (*charstring*, *universal charstring*), and user-defined structured types, particularly *records* which can be used

to define complex types of primitive (or other complex) types, including the structure of the input and output data for the test cases.

- **Test Data**, defined through *templates* and *signatures*: We are particularly interested in templates, which are data values that can be defined in terms of actual values (e.g., string value, integer value), alternative values (e.g., the output of the SUT can be “a” or “b”), unordered lists of values, ranges of numeric values, or string patterns. Templates are expected, and most often used, to define the possible values for test cases’ input values and expected output values (the *send* and *receive* calls in TTCN-3).
- **Test Configurations** defined through special configurations types such as *ports* and *components*: These configurations are necessary for the execution of the test cases in a TTCN-3 environment, and despite the fact that Melba does not execute TCs, these data are used for the creation of new TCs. The port is a communication structure with direction (*in*, *out*, *inout*) and relates the expected input and output data to the component to be used in the test cases. The components are entities on which test behaviours are executed, and may contain declarations and a list of ports to be used.
- **Test Behaviour** defined through *functions*, *named alternative* statements and test cases (*testcase*): The test cases specify the messages to send and receive in each test (by referring to the templates), the computations that should happen, and the assignment of verdicts; they can be decomposed into functions and alternative steps; and they can make use of timers and timeouts.

Module Control: An optional block to control the execution of the test cases in terms of order, repetition, conditions, etc.

Based on those definitions, the following restrictions were established for the use of TTCN-3 in Melba.

1. The use of the structure *record* for the definition of input and output format is required;
2. It is necessary to use a *record* for the input definition and a different one for the output definition, even if they are similar (i.e., same number and/or types of

- parameters), and records are exclusive used for that (i.e., exactly 2 records must be defined in the TS's module);
3. Parameters of compound type (collections) or user-defined type are ignored (i.e., not loaded into the tool);
 4. In the *templates*, for parameter values in intervals (or ranges, e.g. temperature := (15 .. 30)) or list of values (e.g. temperature := (15, 16, 17)), the tool will only obtain the first value for that parameter in that Test Case (in the examples, both cases would result in the value 15 for parameter temperature).
 5. One unique file must be used for the test suite, file (module) import is not supported.
 6. Only the following data types are supported: *boolean*, *integer*, *float*, and *charstring*. The structure *record* must be used for the definition of the parameters and their types (item 1), but a parameter can't be of type *record* in the current version.
 7. Input and output parameters must not have the same name.
 8. A TC name must start with a letter and after that, it may contain numbers and the underline character (), but no symbols, e.g.: \$, ^, %.
 9. A *control* block for the execution of test cases is optional since Melba only needs test data.
 10. The TS must contain a minimum of two test cases exercising two different OECs for each output parameter. This is a restriction of the machine learning algorithm we use: Weka requires that at least two classes, following Weka's terminology (in our context, OECs) be exercised in the data set.

4.4.1.3. Test Suite Parsing

TRex, the TTCN-3 Refactoring and Metrics, open source tool [13], published as an Eclipse plug-in, provides IDE functionality for the TTCN-3 core notation and supports the assessment and automatic restructuring of TTCN-3 test suite files by providing suitable metrics and refactorings. In Melba tool, TRex library was applied to help parsing the structure of the test suite TTCN-3 file, which made possible for Melba tool to automatically obtain both parameters definitions and test cases data from that file.

4.4.2. CP Specification and Abstract Test Suite

It was decided that the expressions the user can provide to specify choices and OECs, should be written in Java in order to be as flexible as a programming language can be.

Janino [33] is an open source embedded Java compiler. It is used in Melba to compile at run time the Java expressions defined by the user for choices and OECs. This allowed Melba to validate their syntax, and then to evaluate them against test cases data, thereby identifying which choices and OECs are involved in test cases.

Recall from section 3.2.1 that Melba should be able to identify cases where more than one choice of a category matches a test case or more than one output equivalence class of an output parameter matches a test case, as well as cases where an input parameter value does not match any of the choices in a specific category, or an output parameter value does not match any OEC for that parameter. This is achieved thanks to Janino. One important issue with the current implementation is that results, i.e., identification of problems in the CP specification, are dependent on the test suite. Indeed, if two choices of a category overlap but no test case in the test suite has data that matches these choices' definitions, then Melba won't be able to detect the problem. Only some kind of theorem proving would help analyze the CP specification without the need for the test suite. However, Melba will identify that choices are not used and this will be reported to the user. Eventually, the user will either identify directly that the choices are erroneous or add test cases which will then allow Melba to identify that two choices match a test case.

4.4.3. Data Model Persistence

SVNKit, [Sub]Versioning for Java [32], is an open source version control system that provides APIs to work with Subversion [31] from Java applications. Its use in the project permits to manage working copies of the Melba data model and to access and manipulate a repository of Melba data models.

Simple, XML Serialization [30], is an open source framework used to serialize (write) and deserialize (read) Java objects to and from an XML file. Its use in Melba provides the persistence of the data model, i.e. recording and recovering Melba objects representing the information in the conceptual models presented in section 4.3. More specifically, information about the CP specification (parameters, categories, choices, choice

definitions, properties and selectors, output equivalence classes) and details about the TTCN-3 test suite (parameters, test cases inputs and outputs) are persisted.

4.4.4. Decision Tree

Weka [14] is an open source software offering a collection of machine learning algorithms for data mining tasks, such as data pre-processing, classification, regression, clustering, association rules, and visualization. It is called directly from the Melba code to produce the decision tree from abstract test cases.

4.4.5. Analysis of DT

SSJ, Stochastic Simulation in Java [19], is an open source library that provides facilities such as computing different measures related to probability distributions, performing goodness-of-fit tests, applying quasi-Monte Carlo methods, collecting statistics. The package used in Melba, called Probability Distributions (probdist), contains a set of Java classes providing methods to compute mass, density, distribution, complementary distribution, and inverse distribution. It was useful to calculate the chi-square distribution for the association between pairs of categories defined in the CP specification but not used in the DT (section 3.4.1.2).

4.5. User Interface Design

While addressing each of the automation requirements listed in section 4.1, the graphical user interface (GUI) design of Melba tool was meant to focus on the simplicity and clarity to perform the necessary tasks. In this section we make frequent references to [15], a recent publication on interface design and excellent source of related principles and maxims as well as some more detailed aspects of GUI design. Two chapters from that reference are used here as the basis for the interface design decisions: Chapter 6 Design Principles [16], and Chapter 10 Interface Components [17].

Melba's GUI was designed to target the following main usability goals in Interface Design [16]:

- *Comprehensibility*: The system interface should be easy to understand, thus the user is able to use it.

- *Learnability*: Learning the process should be straightforward, not implying a high learning curve.
- *Effectiveness/Usefulness*: Actions taken through the system should be effective, which can be achieved with a system that is safe (data and changes to data are recorded and can be recovered), flexible (the actions can be undone if needed, even if this is not fully automated) and stable (actions must perform consistent results), and the available functionalities are clearly useful in the process.
- *Efficiency/Usability*: The user should be able to instantly reach the required tasks which are as simple as possible to perform, easy to memorize, predictable, and clearly visible.

In an attempt to achieve those goals, the GUI was designed as a so-called “WIMP” interface (i.e., comprising Windows, Icons, Menus, and Pointers) [17], as opposed to command line interface for instance. In addition to those backbone widgets, Melba GUI counts on other components, such as buttons, checkboxes, combo boxes, and hyperlinks. Subsections 4.5.1 to 4.5.5 bring a detailed explanation of the interface design decisions in the tool, adding some screenshots to illustrate the accomplishments with each component. These subsections correspond to the sections of [17] that are relevant to the construction of Melba’s GUI, where principles for “WIMP” interface design are systematically discussed.

4.5.1. Document Interface

The strategy chosen here was the Controlled Single Document Interface (CSDI) [17] where only one document (in our context, one SUT) is opened at a time. Therefore any window, tab or menu option available is related to that SUT, and is linked to one primary window (when closed, all linked elements are closed as well). Besides the primary window, other windows support various other tasks and, despite being linked to the primary one, they do not reside within it and have their own separate task bar buttons. This decision to have a single document approach was based on the principle that, in the context of Melba, it is never necessary to compare two different SUT definitions, CP specifications, and results. Having only one SUT at a time avoids confusions. In order to specify the system for the current execution of Melba, the user is first prompted for an

existing SUT or a new one, as illustrated in Figure 11. When switching SUT, by simply loading a new SUT data, all the references to the current SUT are closed.

Figure 11 Choosing SUT to work with at the Melba tool

4.5.2. Windows and Tabs

Practical experience using computing systems shows that the excess of window manipulations can sometimes work against the advantages of a good window-based system, which is summarized by the Maxim “Multiple windows can pose management difficulties” [17]. Therefore opening new windows for new actions was avoided in our GUI design whenever possible. For instance, when defining the CP specification (Figure 12), every step of the specification is done in one unique window, using different tabs for input and output parameters definitions. In addition, inside each tab, the user can easily move from one parameter (or output equivalence class) to another using forward and backward navigations. A new window is only opened when the user executes one of the actions from the application’s menu, which correspond to the main activities of the Melba process, such as executing the machine-learning algorithm and presenting analysis results. The window presenting results has a left part showing the decision tree (among other things) and a right part that helps for the analysis of results: see for instance the top of Figure 13. By moving the vertical separation between these two parts, the user can expand (or shrink) one part for better visualization.

Melba Tool - Taxi_Billing_System

File View Help

CP Specification - Input CP Specification - Output

Step 2 of 2 - Please Provide Equivalence Classes for Output Params

Output Equivalence Classes

Check to be cleared ☐ **Equivalence Class 8**

Total cost is between 0 and 5.80

Parameter	Operation	Between	and
totalCost	exclusive interval	0	5.80

< >

Add more Equivalence Classes Save and validate Clear selected

<< Back to parameters

Figure 12 Specifying category partition (output equivalence class tab active)

This sort of window interface was chosen mainly to provide the tester with a flexible way to visualize two or more windows at the same time only when necessary, as illustrated in Figure 13 where the user can visualize the machine-learning results while applying a modification to the CP Specification. In addition, in the visualization of results the user can also explore the items in the report (right portion of the top window in Figure 13) with the help of control components (hyperlinks) and html tables as explained later in section 4.5.5. Once again, tabs are used (one type of nonconformity per tab).

Figure 13 Visualizing DT Results while modifying CP Specification

4.5.3. Menus and Menu Accelerators

The available menu titles and options follow standard names in contemporary GUIs [17], and comprise: File, View, and Help (Figure 14). It is important for menu titles and options to follow standards because users are familiar to those standards. Being familiar with those standards, the user is able to perceive names as icons which has shown to be useful [17]. Access to the options for all three menus is done either through the cursor (pointing device) or through shortcuts (keyboard accelerators) which are specified in front of the most used options, a useful resource provided for speeding up the process and better assisting experienced users.

The three menus of Melba (Figure 14) are described next in terms of contents and access:

Figure 14 Main menus and options at Melba tool: File (a), View (b), Help (c)

- *File*—Located at the upper left corner of our tool’s main window, the standard File menu is used to group the functions related to the system under test (SUT). This menu all offers all the options to load and manipulate SUT’s data. However, their availability is controlled according to data actually loaded. For instance, in Figure 14, (a) the Taxi Billing System is the current SUT and as data for this system are already loaded, options “Load...” and “Define CP specification...” are unavailable.
- *View*—Located between the other two standard menus, the View menu is used to group the two functions related to the visualization of the SUT’s data: the CP specification which is visualized in a separate window in text format, and the abstract test suite which is generated based on both the loaded TS and the defined CP specification, and is visualized in a separate window in a table format with one ATC per line (Figure 22).
- *Help*—This standard option groups the two functions related to the usage of the tool: “Help Contents” where the user finds guidelines on how to perform the tasks supported by the tool, and “About Melba Tool” that presents a summary of the information about the development of Melba.

4.5.4. Data Entry and Display Components

When defining choices, during CP specification, the user interacts with both radio buttons and checkboxes. Radio buttons are used when the user is only allowed to choose one of a predefined set of options: this is the case for instance when identifying the (only) base choice of a category (Figure 15, second element in Choice 1.1 line). In contrast, checkboxes are used to capture non-mutually exclusive choices: for instance, a checkbox is available next to each choice to clear and remove it from the category (Figure 15, first

element in Choice 1.1 line), thereby allowing the user to make multiple selections for elimination. Capturing Single or Error Choice information also employs checkboxes. Another solution would have been the use of radio buttons because Single and Error options are actually mutually exclusive. However, there is also the possibility for a Choice to be neither Single nor Error, in which case the use of radio buttons would force one of those options to be always selected. Mutual exclusion is illustrated Figure 15 (at the end of the third line of Choice 1.1) where after the selection of Error, the option Single is disabled. De-selecting Error would make both options are enabled again.

When the user can select one element from a set of predefined options, we use comboboxes, a special type of list components. This was particularly useful when presenting predefined operations available for choice or OEC types. The combobox operation in the second line of Choice 1.1 in Figure 15 illustrates that case. We use a listbox, a special kind of combobox, for selecting only one option in a set of mutually exclusive options, as it allows the user to select using either the cursor (pointing device selection) or by typing the first letter of the options, or by using arrow navigation keys.

For the entry of text, as Melba only requires gathering small, discrete amounts of information as opposed to multiple lines text, text boxes are the best choice. These single-line boxes were used to capture information such as descriptions of category, choice and OEC, and definitions of expressions, properties, and selectors. Since these components do not have built-in indicators for indicating their purpose or allowable input, both labels and hints were used to help the user. This can be observed in Figure 15 where labels (e.g., “Choice 1.1”) and hints (e.g., “Pease type a description”) are used in several places.

Figure 15 Labels and hints for text boxes in CP Spec definition. Partial window

Two display components were especially useful in making the interface more flexible and efficient: scrollbars and splitters. Both vertical and horizontal scrollbars were necessary due to the dynamic characteristics of Melba: For instance, when presenting all the choices of a category in the same window (Figure 15, starting from Choice 1.1 for Category 1), or when presenting all OECs in the same window (Figure 17, bottom-positioned at Equivalence Class 8). Similarly, depending on the amount of information to display, horizontal scrollbars may also be used. Another need for using scrollbars in both orientations was in the results window (Figure 23) that provides a complete view of the resulting DT as it may be too long and/or too wide to fit in the space reserved for it. The splitter display component is also used in the results window (Figure 23). The thin vertical bar separating different sections of the results window makes it possible for the user to adjust data visualization according to the focus they need at different moments in the analysis, e.g. moving the splitter to the right in order to focus on DT, or moving it to the left and hiding the DT space in order to have more space and flexibility to analyze and manipulate Nonconformities information.

4.5.5. Control Components

The only two types of control components used in our tool were command buttons and hyperlinks. Command buttons, as the most common type of control components,

were mainly used when defining and saving a category or an OEC and also for navigations between parameter definitions (and their categories) and OECs definitions, the latter being illustrated in Figure 17. Despite buttons being the most common and intuitive component for performing immediate actions, hyperlinks were a better choice in our tool when reporting Nonconformities in the results window. There may be multiple alternative actions available to the user to address each reported nonconformity. In order to organize Nonconformities information, we first used tabs: each nonconformity, i.e., instances of each nonconformity, is presented in its own tab. Then, within tabs, data are presented in HTML tables and the actions available to the user for each of nonconformity are accessible as hyperlinks in the corresponding line of the table. This is illustrated at Figure 16, which shows the tab for reported Unused Combinations of Choices. For each unused combination of choices (first column of the table), the user can expand the line to see more information on that occurrence of nonconformity by activating the combination link in the first column of the table: this is done in Figure 16 for the second instance of unused combination (but not for the first). In the second column of the table, Melba indicates all possible causes for that unused combination, and in the third column Melba displays relevant additional information that could help the user analyze the problem and decide. In the example of Figure 16, the combinations on the second and third lines of the table were expanded, thereby showing each choice of the unused combination. These choices are hyperlinks that can bring the user directly to where those choices are defined in the CP specification. In the second column, no action is available, while in the third column the information “Feasible combination not used in the ATS” is displayed as a hyperlink whose hint shows “Click to define new Test Case”. This link activates the functionality of adding a new test case to the original test suite through the tool. A similar interaction is offered to the user in all of the four nonconformities tabs. Another illustration can be found in Figure 24 for the Misclassifications tab, where beside similar hyperlinks in the first column, misclassified test cases are also displayed as hyperlinks in the second column (to allow the user to explore in details the test case). An important support resource applied with the help of hyperlinks was the use of hints, as cited in the case of defining a new test case. For every single link created in Melba’s GUI, a hint is set either to describe the corresponding action (e.g., to inform that the click on the

nonconformities links will expand that line) or to bring information about that item (e.g., to show test cases' data for a test case link).

Figure 16 Use of Hyperlinks as control components in results tabs. Partial window

4.6. Support Step by Step

This section describes the aspects of the implementation of Melba tool that focuses on making the user's work and decisions easier. In subsection 4.6.1 we discuss pre-defined operations that the user can use when defining the CP specification, and subsection 4.6.2 describes the interface elements in Melba's GUI.

4.6.1. Pre-defined Operations

Allowed parameter (input or output) types in Melba are currently boolean, integer, float, and charstring. For each type Melba provides a set of operations that the user can use when defining parameters' choices and OECs. These operations are summarized in APPENDIX E.

Figure 17 shows an example of how these operations are used. It is the definition of an OEC, for an output parameter `totalCost` of type integer, where the user decided to define an exclusive interval with boundaries 0 and 5.80. This corresponds to the following (Java) expression: $(0 < totalCost) \& \& (totalCost < 5.80)$.

Figure 17 Equivalence Class definition with pre-defined operation *exclusive interval*

4.6.2. Resources in Results GUI

The tool supports user's decision-making process in terms of assessing the potential problems related to test specifications and/or test suites not only by pointing to the potential problems but also by indicating some probable causes for each of problem whenever possible. Figure 18 shows how the decision tree analysis results are presented to the tester.

Figure 18 Complete Melba GUI for DT Analysis

The upper left pane shows the decision tree with the identifications of misclassifications (**M1** and **M2** in the figure) and empty leaves (**E1** and **E2** not visible in the figure—the user should scroll down). Below the decision tree, various classification statistics are also provided (not shown in Figure 18). The lower part of the left pane shows a confusion matrix that summarizes all types of classifications taking place in the decision tree, in addition to a detailed accuracy by class representing several statistics related to the OECs. These two pieces of information (the two parts of the left pane) are similar to what Weka [14] provides.

In the right pane, potential problems are listed in five tabs according to our observations of section 3.4.1. The first tab is a **summary** of all reported observations per type of nonconformity and updatable information on the number of items already analyzed by the test engineer. In each of the next tab, the last column in the table allows the tester to indicate which nonconformity has been investigated (the counts are updated in the “Analyzed” column of the summary tab).

For **misclassifications**, i.e., in the second tab (Figure 24), the leaf identification is provided as well as the number of misclassified test cases, along with possible causes and optional relevant information to investigate. By clicking on the misclassification/leaf identification, the tester will expand that line and see all the CP elements (categories, choices and OEC) composing that branch in the tree, plus the exact list of the test cases misclassified at that branch, and the details of possible causes and relevant information. Besides that, the tester can expand each misclassified test case and visualize all of its input and output data and its actual classification.

For the **unused CP elements**, both choices and OECs that were not used in the decision tree are reported (Figure 16), one per line in a table with their identification, definition, possible causes and relevant information to investigate. By clicking on the identification, the user can see details of that line and even define and create a new test case for the cases where that element is unused because it is missing in the ATS. Figure 26 illustrates the addition of a test case through the tool. It shows in the upper part of the window, at the frame for “Input Data”, that the choice or choices (in case of a missing combination) that are missing are automatically selected in the combo boxes of their corresponding Categories, and for the Categories that are not part of the suggested

combination, the user must select one choice for each to be part of that ATC. The values that the user types for the raw test case creation at the parameters' entry fields are then validated according to the choices selected. The values must all match the choices selected; otherwise the test case cannot be created. The same happens at the bottom frame for "Output Data", with the addition that here, in a case where more than one output parameter is defined for the system, all output parameters (not only the one in use for the current ATS) must be assigned a value, otherwise the (raw) test case would not be complete in the TS file.

The decision tree is also used to identify **empty leaves** which are branches where no instance was classified. The empty leaf identifications are listed one per line in a table with their identification, as well as possible causes and relevant information for investigation. By expanding the line, the tester can see and access every single element composing that empty branch, and also define and create a possibly missing test case by using the same feature described for the unused CP elements.

The last tab brings the occurrences of **unused combinations of choices**, those that are missing in the decision tree. In a similar manner as in the previous tabs, the combinations are listed in addition to possible causes and relevant information to be investigated. And similarly to empty leaves and unused CP elements, in case that combination is not used in the ATS, the tester has the direct option to define and create a new test case for it.

5. CASE STUDY

Having the Melba tool implemented, applying the method becomes easier as less manual activities are necessary and the analysis of possible problems is faster. This case study aims at evaluating how effective the tool is in automating the Melba method. There are three main questions we want to answer:

- I. Does the Melba tool help improve the CP Specification?*
- II. What is the impact the initial TS size has on Melba's result when improving a TS?*
- III. What is the impact the selection of choices has on the method when adding TCs?*

In this case study we rely on the information Melba provides on the "Decision Tree Analysis" window (Section 4.6.2). We will make constant references to the information presented in that interface: mainly the transcript of the Decision Tree on the left, and the Nonconformities tabs on the right of the window, each of them presenting a table with details of each possible problem. We will refer to the columns of those tables containing the information on the automatic analysis of problems as "Relevant Information" and "Possible Causes", as first introduced in Section 4.6.2. We will also refer to the cases where no actual problem exists and no action is necessary as "false positives", as first introduced in Section 3.4.1.3.

Moreover, to evaluate the test suites in each experiment we analyze and compare them in terms of completeness and quality. For completeness, the technique adopted was to measure their code coverage [37]. Cobertura [9] is a free Java tool we chose to evaluate the code coverage of the test cases in each TS. It calculates the percentage of code accessed by the TCs and can be used to identify which parts of the system are lacking test coverage. The system used for the experiments, the Taxi Billing system (Section 5.1.1) includes 159 lines of code and 122 branches.

For the quality aspect, mutation analysis [2, 8] was chosen in order to evaluate how effective the TCs are at detecting faults (mutants) in the system. MuJava [22, 25] was used to handle the Java code of the Taxi Billing system and generate as many mutants as possible. Two types of mutants were applied: the traditional (method-level) ones, and the class-level mutants. From the total of nine classes in the system, only two could not be

handled by MuJava: one for being an Interface (TaxiDao⁷) and one for being an Enumeration (WeekDay). For the other seven classes, a total of 308 traditional mutants and 69 class-level mutants were generated.

The case study is divided in three experiments: one for each of the three evaluation questions. In Section 5.1, the specification of the Taxi Billing system is presented; from Sections 5.2 to 5.4 we describe the experiments, method application, and results for each of our three questions respectively.

5.1. System Specifications

The case study is based on the Taxi Billing System, specified in the next subsection.

5.1.1. The Taxi Billing System

The case study problem is a Taxi Billing System based on the system operating for taxi services in Paris. The system specification was adapted to our study to consider exclusively the charges per kilometre, ignoring the charges per hour that would be applicable when driving the vehicle at lower speed limits. The following rules are applicable:

- Independently from the total amount calculated by the system, the amount to be received by the driver cannot be lower than € 5.80.
- The return trip of the Taxi is never paid by the passenger.
- The starting point of the trip is always considered to be the Urban Area. Therefore, the charge to be used for the calculation is based on the destination area.
- The departure time of the trip is the one used to choose the charge to be applied, even though it can be extended through the next time interval.
- The system calculates the total amount to pay according to Table 5. The initial charge is the value to be added to the total kilometric-based amount to pay.

⁷ This stands for Taxi Data Access Object.

APPLICABLE CHARGES								
Initial Charge: € 2.20	Charge:		A		B		C	
	Per Kilometre		0.86 €/km		1.12 €/km		1.35 €/km	
Time zone	Monday to Saturday					Sunday and Holiday Sunday		Holidays any time
	0 to 7am	7am to 10am	10am to 5pm	5pm to 7pm	7pm to 0	0 to 7am	7am to 0	
Urban Area Paris, including peripheral road	B	B	A	B	B	C	B	B
Suburban Area End of Paris Taxi's area and airport services for Orly, Roissy and Du Parc, and Villepinte expositions	C	B	B	B	C	C	C	C
Beyond Suburban Area	C	C	C	C	C	C	C	C

Table 5 Applicable charges for the Taxi Billing System

The input data for computing the total amount to pay for the trip are: (1) Destination area, (2) Departure time, (3) Day of the week, (4) Whether it is a holiday or not, and (5) Total kilometres run. The output data are: (1) Total cost of the trip and (2) Charge applied (A, B, or C). Error situations are (1) Missing or unknown destination area, (2) Missing or unknown day of the week, (3) Time value less than 0 or greater than 23, and (4) Kilometres run less than or equal to 0. In the event of any of those error situations, the system reports -1 as the total cost of the trip and the charge applied is not informed (empty string).

5.2. Experiment 1

This experiment targets CP Specification improvements by using the Taxi Billing system. It consists of defining a poor CP Specification and a TS with good coverage of the system as initial data for Melba tool, and executing the Melba process to assess the implementation of each of the tool's functional requirements (Table 3). Since this experiment is about improvements through Melba, we needed something to improve upon. We therefore needed a poor CP specification.

In Subsection 5.2.1 we detail the experiment design, including the initial CP Specification and the TS for this experiment; in Subsection 5.2.2 results obtained when applying Melba are described; in Subsections 5.2.3 and 5.2.4 we summarize and discuss the results for the CP Specification and TS, respectively; and in Section 5.2.5 we summarize the functional requirements of Melba we assessed through this experiment.

5.2.1. Experiment Design

The main goal with this experiment is to address our first question; we therefore want to assess the improvements to the CP Specification brought by using Melba tool.

Our steps in the experiment will consist of (1) Loading the ttcn-3 TS file into the tool; (2) Creating and validating the categories and choices for the input parameters and the OECs for the output parameters by using the tool; (3) Generating the ATS and the corresponding DT for all distinct output parameters (in two different executions), and when any problem is found that prevents the generation of these objects, suitable corrections will be applied; (4) Analyzing the reported results based on the DT and initial data, and in case actual problems are identified, suitable corrections will be applied and the system will be re-executed from step 3 until no actual problem is reported or no applicable corrections exist; (5) Analyzing final CP Specification in comparison to the initial one; (6) Analyzing possible changes applied to the initial TS by comparing the percentage of code coverage and mutation scores for the initial and final test suites.

Next, we design the initial CP Specification and the initial TS for this experiment.

5.2.1.1. The CP Specification

In order to define this initial CP Specification, we defined a simple first version for the Taxi Billing system before applying any of Melba's steps. The Categories and Choices, and the Output Equivalence Classes (OECs) were created without considering exclusiveness of expressions or thinking thoroughly about properties and selectors. We even wanted it to be faulty for the purpose of illustrating the detection and resolution of problems with the help of the tool.

This CP Specification is found at Appendix F.1 and it presents five categories, each one defined for each of the five input parameters of the system, and eight OECs, four for each of the two output parameters of the system. The tool requires one of the choices in

each category to be defined as the base choice (Section 4.2), which is indicated in the Specification with an asterisk in front of the base choice and can be used later for a coverage criterion of CP. Category 1 (Area of destination) has four choices, three of them dealing with the possibilities of destination area for this system, and the 4th one dealing with error situations of either this information to be missing or its value unknown. Category 2 (Time of departure) defines eight choices, where two of them deal with error situations for time information being negative or greater than 23 (choices Ch 2.1 and Ch 2.7 respectively). The other six choices define time intervals that are important for determining the charge to be used by the system to calculate the cost of the trip, including the particular case of choice Ch 2.8 that deals with the special departure time interval for Sundays. Category 3 (Day of the week) has three choices where the first one treats the error case when the day of the week is missing or contains unexpected information, while the other two define the cases of Monday to Saturday (choice Ch 3.2) and Sunday (choice Ch 3.3). Category 4 (Holiday) has only two choices, choice Ch 4.1 for holidays and choice Ch 4.2 for non-holidays. Finally, Category 5 (Kilometres run) has two choices as well, where choice Ch 5.1 treats the error situation of value for kilometres run to be illegal, while choice Ch 5.2 encloses the expected positive values for that variable.

For output parameter `totalCost`, the four OECs defined deal with all sorts of values for this parameter, specifying values 0 and 5.80 (minimum cost for a trip) as special values. The last four OECs, for output parameter `chargeApplied`, are based on the possible charges to be applied to a trip according to the system's specifications: A, B, or C (OECs 5, 6, and 7 respectively), and OEC 4 that deals with the error situation of the input data to present problems and no calculation of cost to be done at all, in which case `chargeApplied` is not informed.

5.2.1.2. The Test Suite

The construction of the TS for this experiment went through a series of steps before being defined as the initial well-structured TS. An initial test suite was created by an engineer, knowledgeable in software testing principles (including CP), familiar with the Taxi Billing system, and with industry experience, but without any specific rationale such as applying CP. This initial TS was submitted to the Melba process with a complete CP Specification and improved (addition of test cases) until a point where the DT reported no

Misclassifications for the output parameter `chargeApplied`, and no suggestions of new test cases were left. At that point, only false positives were reported by the tool and the TS was considered complete for use as the initial one in the current experiment. It is detailed in Appendix F.2, and it is composed of 72 test cases.

5.2.2. Applying Melba method

Setup of initial data

Having the initial TS defined and written in the specific TTCN-3 format for Melba tool, the first step was to load the file into the system, and then the taxi billing system parameters and their data types were imported as shown in Figure 19. This illustrates the implementation of functional requirement FR1 (*To load a pre-defined test suite*).

The next step was to define the CP Specification into the tool, by using the interfaces illustrated in Figure 15 for the input data and Figure 17 for the output data: the complete CP Specification provided to the tool can be found in Appendix F.1. This was made possible by implementing functional requirement FR2 (*To load CP Specification from user's definitions*).

Figure 19 Parameters loaded from TS file into Melba Tool

With the input data set up into the tool, the next step was trying to generate the ATS and the corresponding DT. When this action is activated, as the Taxi Billing System defines 2 output parameters, we are prompted to choose one of them for which the ATS should be generated. We will explore this experiment for both output parameters, in 2 different executions.

5.2.2.1.EXECUTION 1 – For output parameter *totalCost*

For the first execution, we choose parameter *totalCost* as illustrated in Figure 20.

Figure 20 Choice of Output Parameter for the ATS

Iteration 1

While generating the ATS, Melba performs some verification in the input data (Section 3.2.1). As a result, the tool informed us that it was not possible to generate the ATS because some input parameter values match different choices in the same category, and some output parameter values match different OECs, as partially illustrated in Figure 21. This early detection of problems in the CP definitions was made possible by implementing functional requirement FR3 (*To generate fully automatically the abstract test suite*).

Figure 21 Error message and list of test case values with multiple matches

In summary, the complete set of pairs with overlapping definitions reported is:

C 2 = Ch 2.8, Ch 2.3

C 2 = Ch 2.8, Ch 2.4

C 2 = Ch 2.8, Ch 2.5

C 2 = Ch 2.8, Ch 2.6

EC 1, EC 2

This prevents the generation of the ATS, therefore at least one element in each of those pairs had to be modified in order to eliminate the multiple matches. By looking at category C 2 we notice that choices 2.3 to 2.6 are conflicting with choice Ch 2.8 because

these choices do not specify that the dayOfWeek must not be Sunday, while for Ch 2.8 it must be Sunday. The corrections applied to that category are in bold as follows:

Parameter: departureTime

Category 2 - Time of departure

Ch 2.1: Negative time [error]

`departureTime < 0`

Ch 2.2: Night

`departureTime >= 0 && departureTime < 7`

Ch 2.3: Morning

`departureTime >= 7 && departureTime < 10 && !dayOfWeek.equals("Sunday")`

*Ch 2.4: Afternoon

`departureTime >= 10 && departureTime < 17 && !dayOfWeek.equals("Sunday")`

Ch 2.5: Early evening

`departureTime >= 17 && departureTime < 19 && !dayOfWeek.equals("Sunday")`

Ch 2.6: Late evening

`departureTime >= 19 && departureTime < 24 && !dayOfWeek.equals("Sunday")`

Ch 2.7: Illegal time [error]

`departureTime >= 24`

Ch 2.8: Sundays special time of departure

`departureTime >= 7 && departureTime < 24 && dayOfWeek.equals("Sunday")`

Output Equivalence Classes EC 1 and EC 2 are in conflict because their expressions overlap for the value of totalCost equals to 5.80. The correction is applied to EC 2 as follows:

Equivalence Class 1: Total cost was less than minimum cost established

`totalCost == 5.80`

Equivalence Class 2: Total cost is greater than minimum cost established

`totalCost > 5.80 (instead of totalCost >= 5.80 as it was before)`

Iteration 2

With these corrections applied to CP Specification of Iteration 1, a second attempt is performed to generate the ATS (still for output parameter totalCost) and the corresponding DT. This time the ATS is successfully generated as shown in Figure 22.

The successful generation of ATS was made possible by implementing functional requirement FR3 (*To generate fully automatically the abstract test suite*).

Test Case	Category 1	Category 2	Category 3	Category 4	Category 5	OEC for: totalCost
test1	Ch 1.1	Ch 2.2	Ch 3.2	Ch 4.2	Ch 5.2	EC 1
test10	Ch 1.1	Ch 2.8	Ch 3.3	Ch 4.2	Ch 5.2	EC 2
test11	Ch 1.1	Ch 2.8	Ch 3.3	Ch 4.1	Ch 5.2	EC 2
test12	Ch 1.2	Ch 2.2	Ch 3.2	Ch 4.2	Ch 5.2	EC 1
test13	Ch 1.2	Ch 2.3	Ch 3.2	Ch 4.2	Ch 5.2	EC 1
test14	Ch 1.2	Ch 2.4	Ch 3.2	Ch 4.2	Ch 5.2	EC 1
test15	Ch 1.2	Ch 2.5	Ch 3.2	Ch 4.2	Ch 5.2	EC 2
test16	Ch 1.2	Ch 2.6	Ch 3.2	Ch 4.2	Ch 5.2	EC 2
test17	Ch 1.2	Ch 2.2	Ch 3.2	Ch 4.1	Ch 5.2	EC 2
test18	Ch 1.2	Ch 2.2	Ch 3.3	Ch 4.2	Ch 5.2	EC 2
test19	Ch 1.2	Ch 2.2	Ch 3.3	Ch 4.1	Ch 5.2	EC 2
test2	Ch 1.1	Ch 2.3	Ch 3.2	Ch 4.2	Ch 5.2	EC 1
test20	Ch 1.2	Ch 2.8	Ch 3.3	Ch 4.2	Ch 5.2	EC 2
test21	Ch 1.2	Ch 2.8	Ch 3.3	Ch 4.1	Ch 5.2	EC 2
test22	Ch 1.3	Ch 2.2	Ch 3.2	Ch 4.2	Ch 5.2	EC 2

Figure 22 ATS for output parameter totalCost (excerpt). TCs sorted alphabetically

The DT built has 23 leaves, as observed in Figure 23. On the left side of Figure 23, one can see the complete DT transcription where some of the leaves appear in bold face: **E** for empty leaves and **M** for misclassifications. This DT results from the implementation of functional requirement FR4 (*To generate automatically the Decision Tree corresponding to the input data*). Shown on the right side of Figure 23, are the 143 Nonconformities that Melba identified from the analysis of the specifications and the DT. By looking at the first group of nonconformities, we have 5 Misclassifications reported, labelled from M1 to M5 in the DT transcription. In Figure 24, we explore the details of the Misclassifications group, by selecting the Misclassifications tab on the right side of Figure 23. On the Relevant Information column of the table in Figure 24, we notice that all the Misclassifications have a choice in common: choice Ch 5.2. Since this could suggest that Ch 5.2 is causing or contributing to the misclassifications, it should be investigated. The identification of Nonconformities and all the detailed information about each of them results from the implementation of functional requirement FR5 (*To support user's decision-making process by reporting every identified results of possible problems, and facilitating the recognition of possible causes*).

In category C 5, choice Ch 5.2 is defined as “Legal distance run” ($\text{kmRun} > 0$), and by looking at the DT again (at the left side of Figure 23), we notice that choices Ch 5.2 and Ch 5.1 ($\text{kmRun} \leq 0$) were used as initial branches in the tree: every TC exercises

either Ch 5.2 or Ch 5.1, meaning that for this DT these two choices work as the first level of distinction between the ATCs in the current ATS. The rule built from choice Ch 5.1 is very simple: C 5 = Ch 5.1: EC 3, and correct according to the system's specifications where all ATCs that exercise this choice are classified as OEC EC 3 (totalCost < 0) as they represent exactly what this OEC is about: "Error in input data, total could not be calculated".

Figure 23 DT and analysis of possible problems by Melba tool

Summary Misclassifications Unused CP Elements Empty Leaves Unused Combinations of Choices				
Leaf with misclassifications	Misclassified TCs	Possible causes	Relevant information	Analyzed
M1 Ch 5.2 Ch 4.2 Ch 3.2 Ch 1.1 EC 1	1 test22	Investigate Missing category or choice Ill-defined choices	Investigate (C 5,Ch 5.2) also used in other branches with misclassifications: M2 M3 M4 M5 (C 2,Ch 2.2) also used in other branches with misclassifications: M2 (C 4,Ch 4.2) also used in other branches with misclassifications: M4	☐
M2 Ch 5.2 Ch 2.2 Ch 3.3 EC 2	1 test29	Investigate Missing category or choice Ill-defined choices	Investigate (C 5,Ch 5.2) also used in other branches with misclassifications: M1 M3 M4 M5 (C 2,Ch 2.2) also used in other branches with misclassifications: M1	☐
M3 Ch 5.2 Ch 2.3 Ch 1.1 EC 2	1 test2	Investigate Missing category or choice Ill-defined choices	Investigate (C 5,Ch 5.2) also used in other branches with misclassifications: M1 M2 M4 M5 (C 1,Ch 1.1) also used in other branches with misclassifications: M4	☐
M4 Ch 5.2 Ch 2.4 Ch 4.2 Ch 1.1 EC 1	1 test58	Investigate Missing category or choice Ill-defined choices	Investigate (C 5,Ch 5.2) also used in other branches with misclassifications: M1 M2 M3 M5 (C 4,Ch 4.2) also used in other branches with misclassifications: M1 (C 1,Ch 1.1) also used in other branches with misclassifications: M3	☐
M5 Ch 5.2 Ch 2.6 EC 2	1 test69	Investigate Missing category or choice Ill-defined choices	Investigate (C 5,Ch 5.2) also used in other branches with misclassifications: M1 M2 M3 M4	☐

Figure 24 Misclassifications for DT of Figure 23

On the other hand, the rule for choice Ch 5.2 is more complex as it contains a series of child branches to decide the OEC for the ATCs that exercise this choice, and as a consequence, the decision for the tree path in that case is made in the next levels of the tree by considering choices in other categories as well. However, as it can be observed again in the DT transcription (left side of Figure 23), the branches starting at Ch 5.2 were built according to the behaviour of most of the ATCs: 52 out of 72 ATCs are used in (and match) these rules. The remaining 5 ATCs that involve Ch 5.2 are misclassified. This shows that the current CP Specification (including OECs) may not be precise enough to permit a distinction among the output equivalence classes defined for the parameter totalCost. This may be due to either problems in the definition of the OECs for totalCost, or in the definition of categories and choices. In this experiment we consider one scenario, which is to update the OECs for totalCost, and the alternative scenario that consists in updating choices' definitions is discussed in Appendix F.4.

From the 4 Output Equivalence Classes defined for totalCost (see below), only the three first were used in the DT since the last one (EC 8) is only possible if there is a fault in the program.

Equivalence Class 1: Total cost was less than minimum cost established

$\text{totalCost} == 5.80$

Equivalence Class 2: Total cost is greater than minimum cost established

$\text{totalCost} > 5.80$

Equivalence Class 3: Error in input data, total could not be calculated

$\text{totalCost} < 0$

Equivalence Class 8: Total cost is between 0 and 5.80

$(0 < \text{totalCost}) \ \&\& \ (\text{totalCost} < 5.80)$

We therefore first group OECs 3 and 8 into one OEC: $\text{totalCost} < 5.80$. Then we consider that it is not necessary to distinguish correct cost values that is to distinguish value 5.80 from values strictly greater than 5.80: we therefore merge OECs 1 and 2 into one OEC $\text{totalCost} \geq 5.80$. The result is the following two OECs:

Equivalence Class 1: Total cost is less than minimum cost established

$\text{totalCost} < 5.80$

Equivalence Class 2: Total cost is greater than or equal to minimum cost established

$\text{totalCost} \geq 5.80$

Turning our attention to the only occurrence of an Unused CP Element (Figure 23), i.e., the unnecessary OEC EC 8 (Total cost is between 0 and 5.80), the Relevant Information shows that this element was not used in DT because it was not even used in ATS (Figure 25). The Possible Causes suggest this element is useless or ill-defined, or that a TC is missing. In the latter case, the user may click on the link “Not used in the ATS” and with the support of the tool, create the necessary TC immediately. However, as we previously analyzed, this OEC actually represents what would be a failure in the implementation of the Taxi Billing System (totalCost calculated between 0 and 5.80); this analysis shows us that it is indeed a useless OEC and we remove it from the CP Specification. These possibilities of correction illustrate the implementation of functional requirement FR6 (*To provide direct application of suggested improvements to both CP*

Specification and TS), that provides similar correction alternatives for all 4 sorts of Nonconformities reported by the tool.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Element	Definition	Possible causes	Relevant information
(EC 8) EC 8	(0 < totalCost) && (totalCost < 5.80)	Investigate Useless equivalence class Ill-defined equivalence class(es) Missing test cases	Investigate <u>Not used in the ATS</u>	

Figure 25 Identification of OEC defined in CP Specification and not used in DT

The Empty Leaves, i.e., E1 [Ch 5.2, Ch 2.4, Ch 4.2, Ch 1.4, EC 1] and E2 [Ch 5.2, Ch 2.5, Ch 1.4, EC 1], are reported as unfeasible by the tool because they both involve Error choice Ch 1.4 (Destination not informed or unknown), which is already used in another ATC. Indeed, an Error choice does not need to be exercised more than once in the ATS⁸. And, if a (error) choice appears in the DT, it is certain that at least one ATC exercising that choice already exists⁹. In that particular case, we are therefore certain that Error choice Ch 1.4 is already involved in at least one ATC. Since they are false positives, no correction is applicable here.

The tool reported 135 Unused Combinations of Choices, from which 128 are false positives: either unfeasible combinations of choices (for involving one or more of the Error choices defined, i.e., Ch 1.4, Ch 2.1, Ch 2.7, Ch 3.1, Ch 5.1, which are already involved in TCs) or combinations already used in the ATS. The other seven are missing combinations suggested for investigation:

1. [(C 5,Ch 5.2) (C 2,Ch 2.8) (C 3,Ch 3.2) (C 4,Ch 4.2)]
2. [(C 5,Ch 5.2) (C 2,Ch 2.8) (C 3,Ch 3.2) (C 4,Ch 4.1)]
3. [(C 5,Ch 5.2) (C 2,Ch 2.3) (C 3,Ch 3.3)]
4. [(C 5,Ch 5.2) (C 2,Ch 2.4) (C 3,Ch 3.3)]
5. [(C 5,Ch 5.2) (C 2,Ch 2.5) (C 3,Ch 3.3)]
6. [(C 5,Ch 5.2) (C 2,Ch 2.6) (C 3,Ch 3.3)]
7. [(C 5,Ch 5.2) (C 2,Ch 2.4) (C 4,Ch 4.1) (C 1,Ch 1.3)]

⁸ Note however, that devising an ATC (and test case) to satisfy these combinations of choices (with the corresponding OEC) is possible. A better wording the tool could have used in these situations is “unnecessary” instead of “un-feasible”.

⁹ Recall that CP Specification elements that are not exercised in the ATS are unknown to the ML algorithm and therefore never used in the DT, which permits to state that any element in the DT was exercised by at least one ATC.

These missing combinations are suggested by the tool as a consequence of either **Problem IV** Missing Test Case, or **Problem V** Ill-defined Properties and/or Selectors (Section 3.5). By looking at the two first cases, it is clear that properties and selectors are really missing in CP Specification since choices Ch 2.8 (Sundays special time of departure) and Ch 3.2 (Weekdays and Saturdays – for dayOfWeek) are not compatible, so these two suggested combinations are not feasible. Property “Weekday” is then added to choice Ch 3.2, and property “Sunday” is added to choice Ch 3.3 (Sunday – for dayOfWeek) to distinguish them. Then, selector “if !Sunday” is added to choices Ch 2.3, Ch 2.4, Ch 2.5, and Ch 2.6 because they all specify departure times that are not valid for Sundays; and selector “if !Weekday” is added to choice Ch 2.8 as it defines an interval of departure time that is exclusive for Sundays. Missing combinations 3 to 6 are similar and actually corrected by the previous change since the combination of any of choices Ch 2.3, Ch 2.4, Ch 2.5, and Ch 2.6 with choice Ch 3.3 is not feasible. Finally, missing combination 7 is a feasible combination not used in the ATS, and by using the tool option for creating a new Test Case (Figure 26), the following combination of elements is added as a new TC (test73):

[(C 5,Ch 5.2) (C 2,Ch 2.4) (C 4,Ch 4.1) (C 1,Ch 1.3)] as suggested by the tool, and choice Ch 3.2 as the option for Category 3 (not suggested in the combination), resulting in the OEC EC 2 (totalCost $\geq$ 5.80).

In Figure 26, as defined at Section 4.6.2, the interface is provided to facilitate the creation of the missing test case and to guarantee that the values will match each CP Spec element that is part of the missing combination. In this case, both output parameters, totalCost and chargeApplied, are assigned values as to complete the TC that is also created physically in the TS file.

Melba Tool - Test Case Definition

Define new Test Case

TC Name:

Input data:

Category C 1 Area of destination:

Parameter destination:

Category C 2 Time of departure:

Parameter departureTime:

Category C 3 Day of the week:

Output data:

Output Equivalence Class:

Parameter totalCost:

Parameter chargeApplied:

☒ load new test cases into system automatically

Figure 26 Definition of new TC through Melba tool

Iteration 3

Using all corrections applied to the CP Specification in Iteration 2, the third iteration is performed. The ATS is generated, still for output parameter `totalCost`, now with one more ATC (and the corresponding concrete test case `test73` in the TTCN-3 TS file). The DT is then built yielding 14 leaves and 64 Nonconformities. As seen in the Summary of Figure 27, the Misclassifications were eliminated as now all the 73 TCs were correctly classified, and no Empty Leaf was found in the tree.

For the 62 Unused Combinations of Choices reported, no further actions are needed as they all represent false positives, where, as checked by the tool, the combinations are either Unfeasible Combinations of Choices or Already Used in the ATS.

Last, there were two Unused CP Elements reported: choices Ch 4.1 and Ch 4.2 which means the whole Category C 4 was left out of the tree. The possible causes indicate that the element may be useless or ill-defined. By performing a test of association for qualitative variables (Section 3.4.1.2) against all the categories in the tree (C 5, C 2, C 3,

C 1), the only suggested possibility of correlation is with C 5. By looking at the ATS, we confirmed that all the TCs exercising choice Ch 5.1 exercise choice Ch 4.2. In order to eliminate this strong association, we changed values for holiday variable in test cases 40 to 42, and 50 to 54 from *false* to *true* (now matching Ch 4.1). Another possibility would be to add new test cases where this association is not true, but we opted for the solution of adapting the existing ones in order to keep the number of test cases smaller and more diverse.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities Identified				
	All	Analyzed		
Misclassifications	0	0		
Unused CP Elements	2	0		
Empty Leaves	0	0		
Unused Combinations of Choices	62	0		
Nonconformities Total	64	0		

Figure 27 Nonconformities identified for Iteration 3 of Execution 1

Iteration 4

Executing Melba steps again brings the same results. The only difference now is that choices in category C 4, still absent from DT, were not found to be correlated with any other category. In theory the fact that a category is not used in a DT is possible if that category (here C 4) does not characterize inputs that can change the value of the selected output parameter (here totalCost). That category may instead characterize inputs that can change the value of another parameter (recall Section 3.4.1.2). This is the case here: category C 4 appears in DTs when we select output parameter chargeApplied (see Table 6 and related discussion). Therefore no further actions are needed for Execution 1.

5.2.2.2.EXECUTION 2 – For output parameter *chargeApplied*

For the second execution, we choose output parameter chargeApplied. Based on the observations of Execution 1, the corrections applied to the CP Specification are pertinent; therefore the initial CP Specification to be used here is the one resulting from Execution 1. And for the TS, up to Iteration 2 all corrections applied are applicable here as well,

thus the TS resulting from Execution 1—Iteration 2 is adopted here, including the newly added test73. The change of values for parameter holiday in some test cases during Iteration 3 is not used here as we want to analyze if the same correlation would be detected in this case as well. This would allow us to illustrate that CP improvements are dependent on the selected OEC. The re-use of the data described, saved in Execution 1, is possible through the implementation of functional requirement FR7 *To save, export, keep tracking and recover system data along the fine tuning process*, which permits the re-use of the definitions in distinct scenarios, such as with a different TS, or a different execution for another output parameter as performed here.

Iteration 1

The first attempt to generate the ATS is successful. It is similar to the ATS generated for the previous execution, except for the last column, which instead of matching values of parameter totalCost to its OECs, now matches values of parameter chargeApplied to its OECs. The complete DT built is listed in Table 6.

A total of 118 Nonconformities were identified, as shown in Figure 28. After investigating, the two Empty Leaves are false positives (Unfeasible combination of choices), and the 116 Unused Combinations of Choices are also false positives (some are reported as Unfeasible combination of choices, others as Already used in the ATS).

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities identified		All	Analyzed	
Misclassifications		0	0	
Unused CP Elements		0	0	
Empty Leaves		2	0	
Unused Combinations of Choices		116	0	
<i>Nonconformities Total</i>		118	0	

Figure 28 Nonconformities identified for Iteration 1 of Execution 2

Based on those results, no correction can be made and therefore, no further iteration is necessary.

J48 unpruned tree

C 5 = Ch 5.2

```

|   C 1 = Ch 1.1
|   |   C 2 = Ch 2.2
|   |   |   C 3 = Ch 3.2: EC 6 (2.0)
|   |   |   C 3 = Ch 3.1: EC 4 (1.0)
|   |   |   C 3 = Ch 3.3: EC 7 (2.0)
|   |   |   C 2 = Ch 2.8: EC 6 (2.0)
|   |   |   C 2 = Ch 2.3: EC 6 (3.0)
|   |   |   C 2 = Ch 2.4
|   |   |   |   C 4 = Ch 4.2: EC 5 (2.0)
|   |   |   |   C 4 = Ch 4.1: EC 6 (2.0)
|   |   |   |   C 2 = Ch 2.7: EC 4 (1.0)
|   |   |   |   C 2 = Ch 2.5: EC 6 (2.0)
|   |   |   |   C 2 = Ch 2.6: EC 6 (2.0)
|   |   |   |   C 2 = Ch 2.1: EC 6 (0.0) E1
|   |   C 1 = Ch 1.2
|   |   |   C 2 = Ch 2.2
|   |   |   |   C 3 = Ch 3.2: EC 7 (2.0)
|   |   |   |   C 3 = Ch 3.3: EC 7 (2.0)
|   |   |   |   C 3 = Ch 3.1: EC 4 (1.0)
|   |   |   |   C 2 = Ch 2.3
|   |   |   |   |   C 4 = Ch 4.2: EC 6 (2.0)
|   |   |   |   |   C 4 = Ch 4.1: EC 7 (2.0)
|   |   |   |   |   C 2 = Ch 2.4
|   |   |   |   |   |   C 4 = Ch 4.2: EC 6 (2.0)
|   |   |   |   |   |   C 4 = Ch 4.1: EC 7 (2.0)
|   |   |   |   |   C 2 = Ch 2.5
|   |   |   |   |   |   C 4 = Ch 4.2: EC 6 (2.0)
|   |   |   |   |   |   C 4 = Ch 4.1: EC 7 (2.0)
|   |   |   |   C 2 = Ch 2.6: EC 7 (2.0)
|   |   |   |   C 2 = Ch 2.8: EC 7 (2.0)
|   |   |   |   C 2 = Ch 2.1: EC 4 (1.0)
|   |   |   |   C 2 = Ch 2.7: EC 4 (2.0)
|   |   C 1 = Ch 1.3
|   |   |   C 2 = Ch 2.2: EC 7 (4.0)
|   |   |   C 2 = Ch 2.3: EC 7 (2.0)
|   |   |   C 2 = Ch 2.4: EC 7 (2.0)
|   |   |   C 2 = Ch 2.5: EC 7 (1.0)
|   |   |   C 2 = Ch 2.6: EC 7 (1.0)
|   |   |   C 2 = Ch 2.8: EC 7 (2.0)
|   |   |   C 2 = Ch 2.1: EC 4 (1.0)
|   |   |   C 2 = Ch 2.7: EC 6 (0.0) E2
|   |   C 1 = Ch 1.4: EC 4 (2.0)
C 5 = Ch 5.1: EC 4 (15.0)

```

Number of Leaves : 34

Size of the tree : 45

Table 6 DT for Iteration 1 - Execution 2

5.2.3. Summarizing Resulting CP Specification

The resulting CP Specification is an improved system specification that, together with the complete TS, produced satisfactory results in both executions, as indicated by the fact that no actual problems were reported in their last iterations. This final CP Specification is listed in Appendix F.3. Following step 5 of this experiment design and comparing it to the initial CP specification, the changes were:

- Choices Ch 2.3, Ch 2.4, Ch 2.5, and Ch 2.6 for departure times on weekdays were made distinguishable from choices for the same departure time on Sundays, by adding the condition `!dayOfWeek.equals("Sunday")` to their expressions;
- Two properties (Sunday and Weekday) were added to choices, as well as five selectors added according to those properties;
- OECs EC 1 and EC 2 were made distinguishable from each other by correcting EC 2 to `totalCost > 5.80` (instead of `totalCost >= 5.80`);
- OEC EC 3 was removed and EC1 and EC2 changed accordingly from EC 1: `totalCost == 5.80`, EC 2: `totalCost > 5.80` to EC 1: `totalCost < 5.80`, EC 2: `totalCost >= 5.80`;
- OEC EC 8 (`((0 < totalCost) && (totalCost < 5.80))`) was found to be useless and it was removed.

5.2.4. Summarizing Resulting Test Suite

The resulting test suite is the original one (TSExec1_Init) with the addition of one test case (test73) which was detected by the tool through a combination of choices that was missing on both DT and ATS:

Input Parameters					Output Parameters	
destination	departureTime	dayOfWeek	holiday	kmRun	totalCost	chargeApplied
test73						
"Beyond suburban" Ch 1.3	16 Ch 2.4	"Tuesday" Ch 3.2	true Ch 4.1	10 Ch 5.2	15.70 EC 2	"C" EC 7

Moreover, value changes for parameter holiday were performed in test cases: 40 to 42, and 50 to 54. These changes to specific test cases were applied to eliminate the correlation between choices of category C 5 and C 4 in the TS when working with output parameter totalCost in Execution 1 (TSExec1_Final). For the output parameter chargeApplied in Execution 2, these changes to holiday values were not used (TSExec2_Final) and proved really unnecessary as no association of categories was

reported. The Coverage percentages and Mutation scores of these three test suites are exactly the same (Table 7).

Test Suite: size	Line Coverage 159 lines	Branch Coverage 122 branches	Traditional Mutants Generated: 308		Class-level Mutants Generated: 69	
	% (lines)	% (branches)	Killed	Total Score %	Killed	Total Score %
TSExec1 Init: 72	98 (156)	86 (106)	232	75.32	23	33.33
TSExec1 Final: 73	98 (156)	86 (106)	232	75.32	23	33.33
TSExec2 Final: 73	98 (156)	86 (106)	232	75.32	23	33.33

Table 7 Comparison of performance of each TS for Experiment 1

It was actually expected not to have improved the initial TS as we started this experiment with a well-structured TS, and it went through very small changes that did not affect the TS effectiveness. However, for the Line Coverage¹⁰ criterion there are three uncovered lines in the code (class TaxiDaoStructuralImpl) that show the possibility for at least three additional test cases that were not identified in the process (Figure 29): lines 61, 70, and 73 are uncovered due to missing TCs for destinationArea "End of area", "Du Parc", and "Villepinte", respectively. They were not spotted by the method because they are all part of the same choice definition, which was exercised in the TS with the other values "Orly" and "Roissy":

Category 1

Ch 1.2: Suburban destination

```
destination.equals("End of area") || destination.equals("Orly") ||
destination.equals("Roissy") || destination.equals("Du Parc") ||
destination.equals("Villepinte"))
```

¹⁰ Line coverage is equivalent to statement coverage and it will be referred here as to line coverage following the term used by Cobertura.

```

43 1
44
45 }
46
47 public ApplicableCharge findApplicableCharge(WeekDay day, boolean holiday,
48 int departureTime, String destinationAreaName) {
49
50     DestinationArea destArea = null;
51     ApplicableCharge ac = null;
52
53     // Destination areas
54     // Urban
55     if (destinationAreaName.equals("Paris")) {
56         destArea = new DestinationArea("Paris", urban);
57     }
58     else if (destinationAreaName.equals("Peripheral road")) {
59         destArea = new DestinationArea("Peripheral road", urban);
60     }
61     // Suburban
62     else if (destinationAreaName.equals("End of area")) {
63         destArea = new DestinationArea("End of area", suburb);
64     }
65     else if (destinationAreaName.equals("Orly")) {
66         destArea = new DestinationArea("Orly", suburb);
67     }
68     else if (destinationAreaName.equals("Roissy")) {
69         destArea = new DestinationArea("Roissy", suburb);
70     }
71     else if (destinationAreaName.equals("De Vinc")) {
72         destArea = new DestinationArea("De Vinc", suburb);
73     }
74     else if (destinationAreaName.equals("Vallepointe")) {
75         destArea = new DestinationArea("Vallepointe", suburb);
76     }
77     // Beyond Suburban
78     else if (destinationAreaName.equals("Beyond suburban")) {
79         destArea = new DestinationArea("Beyond suburban", beyondSuburban);
80     }
81
82     DestinationAreaType destAreaType = null;
83     if (destArea != null) {
84         destAreaType = destArea.getType();
85     }
86
87     //
88     // Applicable Charges
89     //

```

Figure 29 Coverage of TSExec1_Init on the Taxi Billing code. Coverage details on the right, class TaxiDaoStructuralImpl

For the 16 uncovered branches¹¹, all in the same class, three of them are also illustrated in Figure 29 (lines 60, 69, and 72 for the same missing TCs already listed). From the other 13 uncovered branches: six are missing combinations of certain departureTime intervals with the error situation of not informed destinationArea – which does not represent missing TCs since testing an error situation at least once is part of our heuristics (Section 3.4.1.3 Case 3: Missing Combinations of Choices); six are uncovered part(s) of conditions in the same statement. For example, the statement

```
if (departureTime >= 0 && departureTime < 24) {
```

was executed 17 times, evaluated to true 16 times, but only once evaluated to false. This implies one of the conditions in that *if* was evaluated for both true and false, but the other one, only for true – this again does not represent missing test cases because the condition was already tested when evaluating to false and it represents an error situation (*departureTime* either < 0 or ≥ 24); the last uncovered branch is about missing

¹¹ Branch coverage is equivalent to edge coverage and it will be referred here as to branch coverage following the term used by Cobertura.

combinations of certain days of the week from Monday to Saturday with some of the `departureTime` intervals – here again, as days from Monday to Saturday are part of the same input-output relationships in the system, there is no need to have a new TC for every single day from Monday to Saturday with each of the `departureTime` intervals.

Regarding traditional mutation scores, the main types of live (undetected) mutants were:

- change of `departureTime` values with addition, deletion or change of operators (e.g.: *departureTime* changed to *departureTime--*), some not killed due to mutant value fitting in the same interval as actual value, in which interval the program behaviour is the same;
- change of comparison operators (e.g.: (1) checking if the `BigDecimal` number `totalKm` is greater than 0: *if (totalKm.compareTo(new java.math.BigDecimal(0)) == 1)* changed to *if (totalKm.compareTo(new java.math.BigDecimal(0)) >= 1*; or (2) *departureTime >= 19* changed to *departureTime == 19*), some not killed because conditions are still true ((1) method `BigDecimal.compareTo` returns -1, 0, or 1 as this `BigDecimal` is numerically less than, equal to, or greater than the parameter, respectively, and it never returns > 1; (2) the TC exercising that interval uses `departureTime` 19, which still satisfies the condition);
- change of `id` values with addition, deletion or change of operators (e.g.: *id++* changed to *++id* or *id++* changed to *id* , etc. in statements such as: *urban = new DestinationAreaType(id++, "Urban Area");* or *suburb = new DestinationAreaType(id++, "Suburban Area");* or *beyondSuburban = new DestinationAreaType(id++, "Beyond Suburban Area");*) some were not killed because the *id* is currently defined in the class `DestinationAreaType`, but not used in the system;
- inversion of attribute values when creating instance of resulting `ApplicableCharge` (e.g.: *holiday* changed to *!holiday*; *startTime* changed to *~startTime*; *endTime* changed to *~endTime*), not killed because the only output value checked at the applicable charge instance is the *charge* (unchanged).

Finally, the class-level mutants alive were equivalent mutants, and the types fall into:

- *static* is inserted into or deleted from a class attribute declaration, and some of them not killed because there is only one instance of that class in the same execution of the system;
- keyword *this* is added or removed when referring to attributes (e.g.: *this.totalKm* changed to *totalKm* or *this.day* changed to *day* or *totalKm* changed to *this.totalKm* or *day* changed to *this.day*); some were not killed because the attribute is assigned a valid value later on in other methods of the program.

5.2.5. Verification and Validation of the Melba Tool

This experiment shows the tool correctly implements the requirements. During this execution, the Java code coverage tool EclEmma [23] was used to track the percentage of code coverage in the Melba tool, yielding a 77.9% overall coverage: 78% block (or edge) coverage, 77% line (or statement) coverage (6852.2 out of 8949 lines of code), 82% method coverage, and 95% class coverage (161 out of 170 classes)). The code that was not covered falls into several categories:

- Classes, methods and conditional statements for handling exceptions;
- Code implementing functionalities related to the CP specification definition that were not triggered by the experiment as they were not applicable to the specification of the system under test we used, the Taxi Billing System; e.g. the pre-defined operations AllLowerCase (for the verification of all lower-case characters in a string value), and Even and Odd (for the verification of even or odd numeric value) that are offered by the tool when specifying choices;
- Complementary features not exercised, such as adding, removing or renaming a parameter, or removing a category;
- Version Control package, as the execution was performed on the single-file option.

Table 8 summarizes the verification of each functional requirement of the Melba Tool during the execution of Experiment 1.

	Acronym and Description	Verified at
Functional Requirements	FR1 <i>To load a pre-defined test suite</i>	Setup of initial data
	FR2 <i>To load CP Specification from user's definitions</i>	Setup of initial data
	FR3 <i>To generate fully automatically the abstract test suite</i>	Execution 1, Iterations 1 and 2, Execution 2, Iteration 1
	FR4 <i>To generate automatically the Decision Tree corresponding to the input data</i>	Execution 1, Iteration 2
	FR5 <i>To support user's decision-making process by reporting every identified results of possible problems, and facilitating the recognition of possible causes</i>	Execution 1, Iteration 2
	FR6 <i>To provide direct application of suggested improvements to both CP Specification and TS</i>	Execution 1, Iteration 2
	FR7 <i>To save, export, keep tracking and recover system data along the fine tuning process</i>	Setup of Execution 2

Table 8 Requirements verified with the Experiment 1

5.3. Experiment 2

In this experiment we want to evaluate the second aspect proposed: II *What is the impact the initial TS size has on Melba's result when improving a TS?* Specifically, we are interested in how good is a test suite improved through the Melba process, in terms of structural coverage and mutation score, when one starts from initial test suites of varying qualities (in terms of coverage). This question was motivated by the fact that the rules are learned by the DT algorithm through the ATS derived from an initial test suite, which may or may not be “good”, for instance in terms of coverage. Therefore, by trying the Melba method with diverse abstract test suites characterized by different sizes and quality, the TS could have a strong influence on the construction of the DT and the improvement of the TS. We decided to try and measure how much a deficient TS could negatively impact the result of the method, and what would be then the limitations on the achieved results.

In Subsection 5.3.1 we describe the experiment design, including the CP Specification used for this experiment, and the test suites analyzed. In Subsection 5.3.2 we describe the results obtained when applying the Melba process, and in Subsection 5.3.3 the results are summarized.

5.3.1. Experiment Design

Based on Melba results with a base TS, our approach is to create two reduced test suites (reduced from the base one) and apply Melba to them, solving every possible problem reported. The design of the three test suites, one original and two reduced ones,

is detailed in Section 5.3.1.2. At the end of the Melba process, by using a good CP specification (Section 5.3.1.1), when no possible problem is reported or no reported case leads to corrections, we want to compare the resulting test suites to the base one by analyzing how close they are after going through Melba iterations, in terms of code coverage and mutation scores.

5.3.1.1. The CP Specification

As here we are interested in evaluating and measuring the impact of different test suites on the results of the process, the ideal CP Specification to be used in this experiment is the one resulting from the improvements of Experiment 1, found in Appendix F.3.

5.3.1.2. Test Suites

The test suites for the current experiment were defined by analyzing different test suites in terms of completeness and quality by using the techniques defined in Section 5: code coverage and mutation analysis.

Targeting for multiple test suites of different sizes, we decided to use three versions of the same TS: the first one with as close as 100% of line coverage, the second one with about 90% of line coverage, and the third one with about 80% of line coverage. The percentages of coverage are not expected to be lower than those as we want to verify the possibility of reaching the same initial TS by improving the other two through the Melba method. We started from the principle that at Experiment 1 Execution 1 we have a good TS (with 73 TCs) based on the number of TCs and their coverage of code tested, and also based on the most complete DT built, according to the system specification. This initial TS shows good code coverage results (Section 5.2.4), with a small lack of coverage for specific situations (96% overall Line Coverage and 86% overall Branch Coverage). The specific uncovered lines were then manually targeted in the TS by changing the destinationArea from “Orly” to the missing destinations for the following TCs: TC 13 (“End of area”), TC 15 (“Du Parc”), and TC 17 (“Villemontais”), resulting in the TS specified at Appendix F.5, referred here as TS2_1, which assessed for completeness, indicated 100% Line Coverage and 89% Branch Coverage (recall the discussion on

coverage in Section 5.2.4). It shows 75.32% effectiveness by killing 232 traditional mutants in the code.

Next, by randomly removing several TCs and evaluating the completeness and quality of the resulting test suites, we established the other two test suites to be used and evaluated in this experiment. These steps proceeded as follows.

By randomly removing 42 TCs from TS2_1 (Appendix F.6), we ended up with a smaller TS referred here as TS2_2, with 31 TCs, that achieved a line coverage close to our second target of 90%: TS2_2 achieved 88% Line Coverage and 71% Branch Coverage. Its effectiveness was 64.61% by killing 199 traditional mutants.

By randomly removing 13 more TCs from TS2_2 (Appendix F.7), we ended up with the smallest TS referred here as TS2_3, with 18 TCs, that achieved line coverage close to our third target of 80%: TS2_3 achieved 83% Line Coverage and 62% Branch Coverage. Its effectiveness was 59.09% by killing 182 traditional mutants.

For all three test suites, the percentage of class-level mutation score was always 33.33% by killing 23 out of the 69 class-level mutants generated. This was actually expected as according to the analysis of live mutants in Section 5.2.4, the live mutants are equivalent mutants. Therefore this criterion is not used to compare test suites, though we continued to measure it for information purposes. Table 9 summarizes the comparison of these three test suites. Discussion of coverage and mutation rates of those test suites and the improved ones is outlined in Section 5.3.3.

Test Suite: size	Line Coverage 159 lines	Branch Coverage 122 branches	Traditional Mutants Generated: 308		Class-level Mutants Generated: 69	
	% (lines)	% (branches)	Killed	Total Score %	Killed	Total Score %
TS2_1: 73	100 (159)	89 (109)	232	75.32	23	33.33
TS2_2: 31	88 (141)	71 (87)	199	64.61	23	33.33
TS2_3: 18	83 (133)	62 (76)	182	59.09	23	33.33

Table 9 Comparison of performance of each TS for Experiment 2

One thing to note is that for this system, implemented in an Object-Oriented (OO) approach, the code coverage proved to decrease very slowly in comparison to the number of TCs eliminated from each TS and with just a small number of TCs left, the coverage of the TS is still very high. The same occurred with mutation analysis where the number of traditional mutants killed also decreased slowly. This shows that possibly for OO

systems, the code coverage and mutation analysis are not the best options to evaluate test suites, suggesting that other assessment techniques such as data-flow criteria should be investigated.

Having these three test suites defined, the goal was to evaluate them with Melba and through the analysis and reports of the Melba Tool, to try and improve TS2_2 in a first step, and TS2_3 in a second step, and see how close they will be to the original TS2_1 by comparing completeness and effectiveness at the end of the process, and also by comparing redundancy of ATCs.

5.3.1.3. On the Addition of Test Cases when Improving Test Suites

Note that when adding new test cases while trying to improve TS2_2 and TS2_3, we never tried to reuse test cases that we removed when creating these test suites from TS2_1: the removed test cases were randomly selected, and though we could have done so, we never looked at the list of removed test cases when we had to add test cases to TS2_2 and TS2_3. The rationale was to avoid introducing a bias in the experiment: we knew where we came from (TS2_1) but, instead of turning back, we tried to take a different route to hopefully try to reach test suites similar to TS2_1.

One important source of information we used to add test cases is the result of the verification of association between categories (Section 3.4.1.2). During any iteration of the Melba process, several pairs of categories may be identified to be statistically (or practically) associated with one another, thereby suggesting that the test suite does not exercise certain pairs of choices.

Given the set of pairs of associated categories, we followed a greedy algorithm for the addition of test cases that would exercise the missing pairs of choices, thereby trying to add a minimum number of test cases to exercise them. (We did not want to add one new test for each missing pair of choices as this would have likely resulted in too many new test cases.) We first identified the (missing) pairs of choices to be exercised by the new test cases. Then, we proceeded iteratively as follows: (1) We picked a pair not already exercised by one of the new test cases; (2) To create a complete test case specification, we needed to select choices for other categories. We selected choices for these other categories primarily to exercise additional missing pairs. We proceeded this way until no new missing pair could be exercised by the new test case specification, while ensuring

that the test case specification would result in a feasible test case. (3) We repeated the procedure in steps (1) and (2) until all the missing pairs of choices were covered by at least one new test case. This resulted in a many to many relationship between test cases and the missing pairs of choices they exercised: one test case exercising at least one (and potentially more than one) missing pair of choices; one missing pair of choices being exercised by at least one (and potentially more than one) new test case; and missing pairs of choices being exercised by varying numbers of test cases.

Alternatively to the greedy algorithm explained above, we used a second heuristic to add test cases where no missing pairs of choices are identified but a category is still unused in the DT and we have more insight about the unused choices from the system specification. In this case, given a pair of associated categories C_i and C_j where C_i is unused in the DT and C_j is in the DT, we followed a systematic approach for the addition of test cases: (1) For each test case t exercising choice $Ch_{j.x}$ with any of the choices in C_i , we verified according to the system specification whether changing the selected choice in C_i (say using $Ch_{i.2}$ instead of $Ch_{i.1}$) would have impact on the output of that test case (i.e., would change the output equivalence class of t); (2) In case a change in the output would be observed, we added a new test case to the TS, which we call a contrasting TC: a test case exercising almost the same choices as the analyzed test case t , except for the one choice in C_i (using $Ch_{i.2}$ instead of $Ch_{i.1}$) and the output value; (3) We repeated the procedure in steps (1) and (2) until all the choices in C_j were analyzed against all the choices in C_i . This resulted in better variability of the ATS: one original TC where the choice in C_i does not determine the output, and one similar TC where another choice in C_i determines the output, and changes the analysis of the DT algorithm when evaluating correlated categories.

5.3.2. Applying Melba method

For the base test suite, TS2_1, and for output parameter `chargeApplied`, the reported analysis was, as expected, the exact same result as Experiment 1, Execution 2, as presented in Figure 28, therefore raising only false positives: two Empty Leaves (Unfeasible combination of choices), and 116 Unused Combinations of Choices (some are reported as Unfeasible combination of choices, others as Already used in the ATS). By using the same output parameter as in Execution 2, the DT obtained is also the same,

with size 45 (this number is reported by Melba and is the number of nodes in the decision tree) and 34 leaves.

5.3.2.1.Improving TS2_2

For TS2_2 the results were considerably different from the results of the original TS. A total of 50 Nonconformities were identified as shown in Figure 30, and the size of DT is considerably reduced to 15 with 12 leaves.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities identified				
	All		Analyzed	
Misclassifications	5		0	
Unused CP Elements	5		0	
Empty Leaves	1		0	
Unused Combinations of Choices	39		0	
Nonconformities Total	50		0	

Figure 30 Summary of results for TS2_2

Iteration 1

As the 5 Misclassifications reported seem to be due to missing combinations of choices in the DT, we decided to first treat the occurrences that point directly to missing combinations or elements, possibly missing in the ATS as well. The Empty Leaf occurrence is a false positive, an Unfeasible Combination of Choices: E1: [Ch 5.2, Ch 2.4, Ch 1.4, EC 6]. From the 5 Unused CP Elements (choices Ch 3.1, Ch 3.2, Ch 3.3, Ch 4.1, Ch 4.2) only one was not used in the ATS: Ch 3.1 (Day not informed or unknown), and therefore we added TC test1_Scenario2 (Appendix F.8) to the TS. For the other unused choices, no strong association to categories in DT was identified. For the 39 Unused Combinations of Choices, only four are verified by the tool as feasible combinations not used in the ATS. Therefore, four new TCs (test2_Scenario2 to test5_Scenario2) are added (Appendix F.8), and a second iteration is performed.

Iteration 2

This second execution brings 44 Nonconformities as in Figure 31. For both Empty Leaves and Unused Combinations of Choices, there are only false positives reported, as all suggested combinations are verified by the tool as either unfeasible or already used in the ATS.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities identified		All	Analyzed	
Misclassifications		3	0	
Unused CP Elements		5	0	
Empty Leaves		2	0	
Unused Combinations of Choices		34	0	
Nonconformities Total		44	0	

Figure 31 Summary of results for TS2_2 at Iteration 2

The 5 Unused CP Elements were the same as the ones reported in the previous iteration, including Ch 3.1 now exercised by the ATS but still not used in the DT. By looking at the 3 Misclassifications left, we note that the unused choices would be important to be used in the DT as they could avoid those misclassifications¹². Therefore, despite the association numbers calculated by the tool do not indicate that these elements are strongly correlated/associated to the categories in DT, by looking at the ATS, we can see that some of them are actually in a very tight relationship with some other elements in the DT (i.e. the ATCs do not present a good variability of combinations). In other words, though the correlation is not statistically significant, the correlation tables given by the tool show potential, practical correlations. By using this information we observe the category that is more likely to be correlated¹³ with unused C 3 choices: C 2 – from Figure 32¹⁴ we observe a Probability of no association $P = 0.076679$ - and the category that is more likely to be correlated with unused C 4 choices: C 1 - from Figure 33 we observe a

¹² In the branches with misclassifications, if those unused choices were present, the branches would split in more paths and the misclassifications could have been prevented.

¹³ More likely to be correlated is assumed here to be the one with the lowest Probability of no association P .

¹⁴ This figure, as well as all the similar ones that follow, is a screen sample extracted from the Melba tool; the correlation information is presented to the user in the format of a table inside a popup window.

Probability of no association $P = 0.341524$. We then try to give the ATS a better variability of combinations of choices and produce a DT that is more complete.

In Figure 32 choices in C 3 seem to be quite strongly associated to choices in C 2 (though this is not statistically significant). From the ATS, we observe the following points and we apply the respective changes in the TS (new TCs listed at Appendix F.8)¹⁵:

- The two ATCs exercising Ch 2.1 exercise Ch 3.3. Therefore, two TCs were added to the TS combining Ch 2.1 with Ch 3.1: test6_Scenario2, test7_Scenario2; Two TCs were added to the TS combining Ch 2.1 with Ch 3.2: test8_Scenario2, test9_Scenario2.
- From five ATCs exercising Ch 2.2, three exercise Ch 3.3, and two exercise Ch 3.2. Therefore, two TCs were added to the TS combining Ch 2.2 with Ch 3.1: test10_Scenario2, test11_Scenario2.
- The six ATCs exercising Ch 2.3 exercise Ch 3.2. Therefore, two TCs were added to the TS combining Ch 2.3 with Ch 3.1: test12_Scenario2, test13_Scenario2.
- The eight ATCs exercising Ch 2.4 exercise Ch 3.2. Therefore one TC was added to the TS combining Ch 2.4 with Ch 3.1: test14_Scenario2. As Ch 2.4 defines the selector [if !Sunday] and Ch 3.3 defines the property [Sunday], no combination of these choices could be exercised.
- From five ATCs exercising Ch 2.5, four exercise Ch 3.2, and one exercises Ch 3.1. No TCs were added to the TS. Ch 2.5 also defines the selector [if !Sunday] preventing it to be combined with Ch 3.3.
- The five ATCs exercising Ch 2.6 exercise Ch 3.2. Therefore, one TC was added to the TS combining Ch 2.6 with Ch 3.1: test15_Scenario2. Ch 2.6 also defines the selector [if !Sunday] preventing it to be combined with Ch 3.3.
- The one ATC exercising Ch 2.7 exercises Ch 3.2. Therefore, one TC was added to the TS combining Ch 2.7 with Ch 3.1: test16_Scenario2. One TC was added to the TS combining Ch 2.7 with Ch 3.3: test17_Scenario2.

¹⁵ Recall the heuristic we applied to add test cases: Section 5.3.1.3; that explains the varying number of new test cases.

- The four ATCs exercising Ch 2.8 exercise Ch 3.3. Therefore, one TC was added to the TS combining Ch 2.8 with Ch 3.1: test18_Scenario2. As Ch 2.8 defines the selector [if !Weekday] and Ch 3.2 defines the property [Weekday], no combination of these choices could be exercised.

		C 2							
C 3		Ch 2.1	Ch 2.2	Ch 2.3	Ch 2.4	Ch 2.5	Ch 2.6	Ch 2.7	Ch 2.8
	Ch 3.1	-0.235	-0.431	-0.342	-0.592	2.442	-0.431	-0.235	-0.342
	Ch 3.2	-1.567	-1.503	0.976	1.690	-0.137	1.230	0.672	-2.277
	Ch 3.3	1.777	1.808	-0.861	-1.491	-1.085	-1.085	-0.592	2.582

Total test statistic (Chi Squared approximation): 22.095 with 14 degrees of freedom
Probability of no association* P = 0.076679

Figure 32 Correlation calculation for choices in categories C 3 and C 2, using TS2_2

In Figure 33 choices in C 4 seem to be quite strongly associated to choices in C 1, though this is not statistically significant. From the ATS, we observe that:

- From ten ATCs exercising Ch 1.1, seven exercise Ch 4.2, and three exercise Ch 4.1. Therefore six TCs were added to the TS combining Ch 1.1 with Ch 4.1: test6_Scenario2, test7_Scenario2, test8_Scenario2, test16_Scenario2, test19_Scenario2, test21_Scenario2. One TC was added to the TS combining Ch 1.1 with Ch 4.2: test20_Scenario2.
- From 13 ATCs exercising Ch 1.2, nine exercise Ch 4.1, and four exercise Ch 4.2. Therefore, five TCs were added to the TS combining Ch 1.2 with Ch 4.2: test9_Scenario2, test10_Scenario2, test11_Scenario2, test12_Scenario2, test17_Scenario2.
- From eleven ATCs exercising Ch 1.3, seven exercise Ch 4.2, and four exercise Ch 4.1. Therefore, three TCs were added to the TS combining Ch 1.3 with Ch 4.1: test13_Scenario2, test15_Scenario2, test18_Scenario2.
- From two ATCs exercising Ch 1.4, one exercises Ch 4.2, and one exercises Ch 4.1. Therefore, one TC was added to the TS combining Ch 1.4 with Ch 4.1: test14_Scenario2.

C 4	C 1			
	Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4
	Ch 4.1	Ch 4.2	Ch 4.3	Ch 4.4
	-0.724	1.708	-1.222	0.197
	0.724	-1.708	1.222	-0.197

Total test statistic (Chi Squared approximation): 3.344 with 3 degrees of freedom
 Probability of no association* P = 0.341524

Figure 33 Correlation calculation for choices in categories C 4 and C 1, using TS2_2

Iteration 3

This third execution brings 109 Nonconformities as in Figure 34, and the DT has now significantly changed: size of DT is 36 with 29 leaves. All CP elements defined were used in the DT.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities identified				
	All	Analyzed		
Misclassifications	5	0		
Unused CP Elements	0	0		
Empty Leaves	8	0		
Unused Combinations of Choices	96	0		
<i>Nonconformities Total</i>	109	0		

Figure 34 Summary of results for TS2_2 at Iteration 3

From eight Empty Leaves, seven are automatically checked as unfeasible combinations and one is suggested as a possibly missing combination of choices E7: [Ch 3.2, Ch 1.1, Ch 2.2, EC 4], however it is manually checked as unfeasible as well, due to the OEC EC 4 (chargeApplied is Empty) that cannot be an output for this combination of choices. And from 96 Unused Combinations of Choices, only six are automatically checked as feasible combinations, and are therefore added as TCs test22_Scenario2 to test27_Scenario2 (Appendix F.8).

Iteration 4

This time the execution brings 108 Nonconformities as in Figure 35, and the DT keeps its previous size of 36 with 29 leaves. No missing combination of choices is

suggested from the seven Empty Leaves (they are all unfeasible combinations) or from the 96 Unused Combinations of Choices (some are unfeasible, and others are already used in the ATS).

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities Identified	All		Analyzed	
Misclassifications	5		0	
Unused CP Elements	0		0	
Empty Leaves	7		0	
Unused Combinations of Choices	96		0	
<i>Nonconformities Total</i>	108		0	

Figure 35 Summary of results for TS2_2 at Iteration 4

The five misclassified ATCs occur in four different branches in the DT (Figure 36): the actual OECs for the misclassified ATCs were EC 6 for test11 and test14, EC 4 for test52 and test48, and EC 7 for test17. The suggested possible causes, Missing category or choice or Ill-defined choices, are not applicable to these cases as our CP specification is considered complete and correct (no missing or ill-defined choices) as of our results from Experiment 1. With no further actions to be taken, the TS with 58 TCs at Iteration 4 is the Melba result for TS2_2.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Leaf with misclassifications	Misclassified TCs	Possible causes	Relevant information	
M1 Ch 3.3 Ch 2.8 EC 7	1 test11	Investigate Missing category or choice Ill-defined choices	Investigate Unknown	
M2 Ch 3.2 Ch 1.2 Ch 2.4 EC 7	2 test14 test52	Investigate Missing category or choice Ill-defined choices	Investigate (C 3, Ch 3.2) also used in other branches with misclassifications: M3 M4 (C 1, Ch 1.2) also used in other branches with misclassifications: M3	
M3 Ch 3.2 Ch 1.2 Ch 2.2 EC 4	1 test17	Investigate Missing category or choice Ill-defined choices	Investigate (C 3, Ch 3.2) also used in other branches with misclassifications: M2 M4 (C 1, Ch 1.2) also used in other branches with misclassifications: M2	
M4 Ch 3.2 Ch 1.1 Ch 2.5 EC 6	1 test48	Investigate Missing category or choice Ill-defined choices	Investigate (C 3, Ch 3.2) also used in other branches with misclassifications: M2 M3	

Figure 36 Misclassifications for TS2_2 at Iteration 4

5.3.2.2.Improving TS2_3

Starting the Melba process for TS2_3, the results present a total of 26 Nonconformities as shown in Figure 37. The DT has size 9 with only 6 leaves.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities identified		All	Analyzed	
Misclassifications		4	0	
Unused CP Elements		11	0	
Empty Leaves		0	0	
Unused Combinations of Choices		11	0	
Nonconformities Total		26	0	

Figure 37 Summary of results for TS2_3

Iteration 1

Starting from the suggested missing test cases, we first look at the eleven Unused CP Elements where only two are not used in the ATS: Ch 2.1 and Ch 3.1. Test case test1_ts2_3 (Appendix F.9) is added to exercise these choices. From the eleven Unused Combinations of Choices, only one is verified by the tool as feasible, and added to the TS as test2_ts2_3 (Appendix F.9).

Iteration 2

Figure 38 shows the results of iteration 2: 29 Nonconformities with a DT of size 10 with 7 leaves.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities identified				
	All		Analyzed	
Misclassifications	4		0	
Unused CP Elements	10		0	
Empty Leaves	0		0	
Unused Combinations of Choices	15		0	
<i>Nonconformities Total</i>	29		0	

Figure 38 Summary of results for TS2_3, Iteration 2

The 15 Unused Combinations of Choices are checked as unfeasible combinations or already used in the ATS. From the ten Unused CP Elements, eight are the choices from category C 2 (Time of departure), and two are the choices from category C 4 (Holiday). As these two categories were completely left out of the DT, this may have happened due to a strong association of these with other categories in the DT. Therefore, despite the association numbers calculated by the tool that do not indicate that these elements are statistically strongly correlated / associated to other categories in the DT, by looking at the ATS we can see that the ATCs do not present a good variability of combinations. Also, by using the correlation information given by the tool we observe the category that is more likely to be correlated¹³ with unused C 2 is C 1 – from Figure 39 we observe a Probability of no association $P = 0.301450$ - and the category that is more likely to be correlated with unused C 4 is C 1 – from Figure 40 we observe a Probability of no association $P = 0.530777$. This is confirmed by a visual inspection of the ATS. We then try to give the ATS a better variability of combinations and produce a DT that is more complete.

In Figure 39 choices in C 2 do not seem to be very well combined to choices in C 1 as some possible combinations of these choices are not exercised. From the ATS, we

observe the following points and we apply the respective changes in the TS (new TCs listed at Appendix F.9)¹⁵:

- From seven ATCs exercising Ch 1.1, one exercises Ch 2.1, one exercises Ch 2.2, two exercise Ch 2.4, one exercises Ch 2.5, one exercises Ch 2.7, one exercises Ch 2.8. There is no combination with Ch 2.3, or Ch 2.6. Therefore one TC was added to the TS combining Ch 1.1 with Ch 2.3: test3_ts2_3, and one TC was added to the TS combining Ch 1.1 with Ch 2.6: test4_ts2_3.
- From six ATCs exercising Ch 1.2, two exercise Ch 2.2, two exercise Ch 2.4, one exercises Ch 2.6, one exercises Ch 2.8. There is no combination with Ch 2.1, Ch 2.3, Ch 2.5, or Ch 2.7. Therefore, one TC was added to the TS combining Ch 1.2 with Ch 2.1 (test5_ts2_3), Ch 1.2 with Ch 2.3 (test6_ts2_3), Ch 1.2 with Ch 2.5 (test7_ts2_3), Ch 1.2 with Ch 2.7 (test8_ts2_3).

		C 1			
		Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4
C 2	Ch 2.1	1.232	-0.665	-0.572	-0.258
	Ch 2.2	-0.304	0.164	0.441	-0.477
	Ch 2.3	-0.862	-0.665	-0.572	4.123
	Ch 2.4	-0.064	0.618	-0.221	-0.665
	Ch 2.5	0.270	-0.972	0.940	-0.376
	Ch 2.6	-0.862	1.597	-0.572	-0.258
	Ch 2.7	1.232	-0.665	-0.572	-0.258
	Ch 2.8	-0.304	0.164	0.441	-0.477

Total test statistic (Chi Squared approximation): 23.828 with 21 degrees of freedom
Probability of no association* P = 0.301450

Figure 39 Correlation calculation for choices in categories C 2 and C 1, using TS2_3

- From six ATCs exercising Ch 1.3, one exercises Ch 2.2, one exercises Ch 2.3, one exercises Ch 2.4, one exercises Ch 2.5, one exercises Ch 2.6, one exercises Ch 2.8. There is no combination with Ch 2.1 or Ch 2.7. Therefore, one TC was added to the TS combining Ch 1.3 with Ch 2.1 (test9_ts2_3), and one TC was added to the TS combining Ch 1.3 with Ch 2.7 (test10_ts2_3).

C 4	C 1			
	Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4
	Ch 4.1	Ch 4.2	Ch 4.3	Ch 4.4
	-0.485	0.262	-0.493	1.396
	0.485	-0.262	0.493	-1.396

Total test statistic (Chi Squared approximation): 2.206 with 3 degrees of freedom
 Probability of no association* P = 0.530777

Figure 40 Correlation calculation for choices in categories C 4 and C 1, using TS2_3

In Figure 40 the choices in C 4 seem to be quite strongly associated to choices in C 1. From the ATS, we observe the following and we apply the respective changes in the TS (some changes are in common with the ones applied for choices in category C 2):

- From seven ATCs exercising Ch 1.1, two exercise Ch 4.1, and five exercise Ch 4.2. Therefore, two TCs were added to the TS combining Ch 1.1 with Ch 4.1 (test3_ts2_3, test4_ts2_3).
- From six ATCs exercising Ch 1.2, three exercise Ch 4.1, and three exercise Ch 4.2. Therefore, two TCs were added to the TS combining Ch 1.2 with Ch 4.1: test5_ts2_3, test7_ts2_3; and two TCs were added to the TS combining Ch 1.2 with Ch 4.2: test6_ts2_3, test8_ts2_3.
- From six ATCs exercising Ch 1.3, two exercise Ch 4.1, and four exercise Ch 4.2. Therefore, two TCs were added to the TS combining Ch 1.3 with Ch 4.1: test9_ts2_3, test10_ts2_3.

Iteration 3

Figure 41 shows 44 Nonconformities reported, with the DT of size 13, and 11 leaves. The Empty Leaf is verified by the tool as unfeasible, as well as six of the 28 Unused Combinations of Choices, while the other 22 are feasible combinations already used in the ATS.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities Identified		All	Analyzed	
Misclassifications		8	0	
Unused CP Elements		7	0	
Empty Leaves		1	0	
Unused Combinations of Choices		28	0	
Nonconformities Total		44	0	

Figure 41 Summary of results for TS2_3, Iteration 3

As misclassifications seem hard to solve at this point (looking at the branches of the tree does not bring any intuition as to the root cause of misclassifications), we turn our attention to the seven Unused CP Elements, which represent all the choices in categories C 3, C 4, and C 5. By looking at the calculation of correlation of these unused elements, only C 3 presents a strong association with a category from the DT: C 2 – from Figure 42 we observe a Probability of no association $P = 0.008627$ (which is statistically significant, suggesting there is a strong association).

C 3	C 2								
		Ch 2.1	Ch 2.2	Ch 2.3	Ch 2.4	Ch 2.5	Ch 2.6	Ch 2.7	Ch 2.8
	Ch 3.1	2.940	-0.416	-0.416	-0.475	-0.353	-0.353	-0.353	-0.353
	Ch 3.2	-0.532	-1.504	1.128	1.288	0.957	0.957	0.957	-3.510
	Ch 3.3	-0.855	1.813	-1.007	-1.150	-0.855	-0.855	-0.855	3.931

Total test statistic (Chi Squared approximation): 29.612 with 14 degrees of freedom
 Probability of no association* $P = 0.008627$ Suggests C 3 and C 2 are strongly correlated

Figure 42 Correlation calculation for choices in categories C 3 and C 2, using TS2_3

Based on the information from Figure 42 and by examining the ATS, we observe the following points and we apply the respective changes in the TS (new TCs listed at Appendix F.9):

- From three ATCs exercising Ch 2.1, one exercises Ch 3.1, and two exercise Ch 3.2. There is no combination of Ch 2.1 and Ch 3.3. One TC was added to the TS combining Ch 2.1 with Ch 3.3: test11_ts2_3.

- From four ATCs exercising Ch 2.2, two exercise Ch 3.2, and two exercise Ch 3.3. There is no combination of Ch 2.2 and Ch 3.1. Therefore one TC was added to the TS combining Ch 2.2 with Ch 3.1: test12_ts2_3.
- The four ATCs exercising Ch 2.3 exercise Ch 3.2. There is no combination of Ch 2.3 with Ch 3.1, or Ch 3.3. Therefore, one TC was added to the TS combining Ch 2.3 with Ch 3.1 (test13_ts2_3). As Ch 2.3 defines the selector [if !Sunday] and Ch 3.3 defines the property [Sunday], no combination of these choices is possible.
- The five ATCs exercising Ch 2.4 exercise Ch 3.2. There is no combination of Ch 2.4 with Ch 3.1, or Ch 3.3. Therefore, one TC was added to the TS combining Ch 2.4 with Ch 3.1: test14_ts2_3. As Ch 2.4 defines the selector [if !Sunday], no combination with Ch 3.3 is possible.
- The three ATCs exercising Ch 2.5 exercise Ch 3.2. No combination of Ch 2.5 with Ch 3.1, or Ch 3.3 is exercised. Therefore, one TC was added to the TS combining Ch 2.5 with Ch 3.1: test15_ts2_3. As Ch 2.5 defines the selector [if !Sunday], no combination with Ch 3.3 is possible.
- The three ATCs exercising Ch 2.6 exercise Ch 3.2. No combination of Ch 2.6 with Ch 3.1, or Ch 3.3 is exercised. Therefore, one TC was added to the TS combining Ch 2.6 with Ch 3.1: test16_ts2_3. As Ch 2.6 defines the selector [if !Sunday], no combination with Ch 3.3 is possible.
- The three ATCs exercising Ch 2.7 exercise Ch 3.2. No combination of Ch 2.7 with Ch 3.1, or Ch 3.3 is exercised. Therefore, one TC was added to the TS combining Ch 2.7 with Ch 3.1 (test17_ts2_3); one TC was added to the TS combining Ch 2.7 with Ch 3.3 (test18_ts2_3).
- The three ATCs exercising Ch 2.8 exercise Ch 3.3. No combination of Ch 2.8 with Ch 3.1, or Ch 3.2 is exercised. No TCs were added to the TS: as Ch 2.8 defines selector [if !Weekday] and Ch 3.2 defines property [Weekday], they could not be combined; and as Ch 2.8 (Sundays special time of departure) defines in the choice expression "... && dayOfWeek.equals("Sunday")", no combination with Ch 3.1 (Day not informed or unknown), with expression "... && !dayOfWeek.equals("Sunday")", is possible.

Iteration 4

Figure 43 shows 75 Nonconformities reported, with the DT of size 18, and 14 leaves. The Empty Leaf is verified by the tool as unfeasible, as well as 13 from the 63 Unused Combinations of Choices, while the other 50 are feasible combinations already used in the ATS.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities Identified	All		Analyzed	
Misclassifications	9		0	
Unused CP Elements	2		0	
Empty Leaves	1		0	
Unused Combinations of Choices	63		0	
<i>Nonconformities Total</i>	75		0	

Figure 43 Summary of results for TS2_3, Iteration 4

The two Unused CP Elements are the choices in category C 4 (Holiday). The nine Misclassifications are located in seven different branches as illustrated in Figure 44. The misclassifications indicate as possible causes that the CP Specification may miss category or choices or have ill-defined ones, but this is not the case. However, as we observe in the DT, the branches with misclassifications do not include all necessary choices to correctly classify those TCs, e.g.: test64 (expected output: EC 7 “Charge C”, instead of EC 6 “Charge B”) could have been correctly classified if branch M4 (Figure 44) had involved choices in C 4 (Holiday). As choices in C 4 were not used in any branch of the DT, we will try to solve that (simple) problem first. We discuss in Section 5.3.3 the order in which we address the nonconformities.

Mella Tool - Decision Tree Analysis				
Decision Tree				
J48 unpruned tree				
C 2 = Ch 2.7: EC 4 (5.0)				
C 2 = Ch 2.8: EC 7 (3.0/1.0) M1				
C 2 = Ch 2.1: EC 4 (4.0)				
C 2 = Ch 2.2				
C 3 = Ch 3.1: EC 4 (1.0)				
C 3 = Ch 3.2: EC 4 (2.0/1.0) M2				
C 3 = Ch 3.3: EC 7 (2.0)				
C 2 = Ch 2.3				
C 5 = Ch 5.1: EC 4 (2.0)				
C 5 = Ch 5.2: EC 6 (3.0/1.0) M3				
C 2 = Ch 2.4				
C 1 = Ch 1.2: EC 6 (2.0/1.0) M4				
C 1 = Ch 1.1: EC 5 (3.0/1.0) M5				
C 1 = Ch 1.3: EC 7 (1.0)				
C 1 = Ch 1.4: EC 4 (0.0) M1				
C 2 = Ch 2.5: EC 7 (4.0/2.0) M6				
C 2 = Ch 2.6: EC 4 (4.0/2.0) M7				
Number of Leaves : 14				
Size of the tree : 10				
--- Summary ---				
Correctly Classified Instances				
Incorrectly Classified Instances				
Kappa statistic				
Mean absolute error				
Root mean squared error				
Relative absolute error				
Root relative squared error				
Total Number of Instances				
Summary Misclassifications Unused CP Elements Empty Leaves Unused Combinations of Choices				
Leaf with misclassifications	Misclassified TCs	Possible causes	Relevant information	
M1 Ch 2.8 EC 7	1 test11	Investigate Missing category or choice Ill-defined choices	Unknown	Investigate
M2 Ch 2.2 Ch 3.2 EC 4	1 test17	Investigate Missing category or choice Ill-defined choices	Unknown	Investigate
M3 Ch 2.3 Ch 5.2 EC 6	1 test34	Investigate Missing category or choice Ill-defined choices	Unknown	Investigate
M4 Ch 2.4 Ch 1.2 EC 6	1 test64	Investigate Missing category or choice Ill-defined choices	(C 2, Ch 2.4) also used in other branches with misclassifications: M5	Investigate
M5 Ch 2.4 Ch 1.1 EC 5	1 test14 test 3	Investigate Missing category or choice Ill-defined choices	(C 2, Ch 2.4) also used in other branches with misclassifications: M4	Investigate
M6 Ch 2.5 EC 7	2 test15 test 3 test4	Investigate Missing category or choice Ill-defined choices	Unknown	Investigate
M7 Ch 2.6 EC 4	2 test16 test4 test 3	Investigate Missing category or choice Ill-defined choices	Unknown	Investigate

Figure 44 DT and Misclassifications at Iteration 4, with TS2_3

By observing the correlation information provided, C 4 is more likely to be correlated¹³ to C 3 though this is not statistically significant – from Figure 45 we observe a Probability of no association $P = 0.082085$. In the ATS, the following points are observed and we apply the respective changes in the TS (new TCs listed at Appendix F.9)¹⁵:

		C 3		
C 4		Ch 3.1	Ch 3.2	Ch 3.3
	Ch 4.1	0.913	1.491	-2.236
	Ch 4.2	-0.913	-1.491	2.236

Total test statistic (Chi Squared approximation): 5.000 with 2 degrees of freedom
Probability of no association* $P = 0.082085$

Figure 45 Correlation calculation for choices in categories C 4 and C 3, using TS2_3

- From seven ATCs exercising Ch 3.1, three exercise Ch 4.1, and four exercise Ch 4.2. As we want to add ATCs where choices in C 4 influence in the output of the ATCs, no TC is added here because Ch 3.1 (Day not informed or unknown) is an Error choice that will lead the output to OEC EC 4 no matter what other choices selected.
- From 22 ATCs exercising Ch 3.2, 12 exercise Ch 4.1, and ten exercise Ch 4.2. From these 22 ATCs, nine have the OEC EC 4 (Error in input data, charge was not used)

- and therefore no matter what choice in C 4 is selected, the OEC is the same. For the other 13:
- Two ATCs result in EC 5 (Charge A), and the choice selected in C 4 is Ch 4.2 (false). Therefore, two ATCs are added with Ch 4.1 selected, resulting in EC 6: test19_ts2_3 and test20_ts2_3 (contrasting¹⁶ test3 and test58 respectively).
 - Five ATCs result in EC 6 (Charge B), and the choices selected in C 4 are Ch 4.1 (true) for two ATCs, and Ch 4.2 for three ATCs. Therefore, two ATCs are added with Ch 4.1 selected, resulting in EC 7: test21_ts2_3 and test22_ts2_3 (contrasting test14 and test6_ts2_3 respectively).
 - Six ATCs result in EC 7 (Charge C), and the choices selected in C 4 are Ch 4.1 for three ATCs, and Ch 4.2 for three ATCs. Therefore, two ATCs are added with Ch 4.2 selected, resulting in EC 6: test23_ts2_3 and test24_ts2_3 (contrasting test64 and test7_ts2_3 respectively).
 - From seven ATCs exercising Ch 3.3, three exercise Ch 4.1, and four exercise Ch 4.2. As we want to add ATCs where choices in C 4 influence in the output of the ATCs, no TC is added here because Ch 3.3 (Sundays) will lead the output to either OEC EC 6 or EC 7 according to the destination area, no matter what choice is selected for C 4.

Iteration 5

At Figure 46, 48 Nonconformities are reported and the DT has size 33 with 27 leaves. The Misclassifications involve five ATCs in four different branches, and do not indicate any actual problem with definitions in the CP Specification. The five Unused CP Elements are now the choices in C 3 (Day of the week) and C 5 (Kilometres run), and no strong correlation with categories in the DT are measured. From the 38 Unused Combinations of Choices, 30 are unfeasible combinations or feasible but already used in the ATS, but eight are verified by the tool as feasible combinations not used in the ATS. As these missing combinations could affect the DT, we first focus on adding them: TCs from test25_ts2_3 to test32_ts2_3 added (Appendix F.9).

¹⁶ We reused the combinations of choices in those test cases and replaced occurrences of Ch 4.2 with Ch 4.1. We also updated the OEC accordingly.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities identified				
	All	Analyzed		
Misclassifications	5	0		
Unused CP Elements	5	0		
Empty Leaves	0	0		
Unused Combinations of Choices	38	0		
<i>Nonconformities Total</i>	48	0		

Figure 46 Summary of results for TS2_3, Iteration 5

Iteration 6

At Figure 47, 49 Nonconformities are reported and the DT has size 33 with 27 leaves. All of the 38 Unused Combinations of Choices are now verified by the tool as unfeasible combinations or feasible but already used in the ATS.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities identified				
	All	Analyzed		
Misclassifications	6	0		
Unused CP Elements	5	0		
Empty Leaves	0	0		
Unused Combinations of Choices	38	0		
<i>Nonconformities Total</i>	49	0		

Figure 47 Summary of results for TS2_3, Iteration 6

The misclassifications still occur in four different branches, but now involving six ATCs. The five Unused CP Elements are still the choices in C 3 and C 5, and this time C 3 correlation calculations point to a second possibly strong association to C 2 (Time of departure), from Figure 48 we observe a Probability of no association $P = 0.002951$ – note that a strong association of C 3 with C 2 was already spotted and addressed during Iteration 3 (Figure 42).

		C 2							
C 3		Ch 2.1	Ch 2.2	Ch 2.3	Ch 2.4	Ch 2.5	Ch 2.6	Ch 2.7	Ch 2.8
	Ch 3.1	0.767	0.131	0.307	-0.295	0.307	0.307	-0.685	-0.905
	Ch 3.2	-0.811	-1.552	0.858	1.676	0.858	0.858	1.226	-3.455
	Ch 3.3	0.311	1.728	-1.276	-1.735	-1.276	-1.276	-0.871	4.860

Total test statistic (Chi Squared approximation): 32.928 with 14 degrees of freedom
 Probability of no association* P = 0.002951 Suggests C 3 and C 2 are strongly correlated

Figure 48 Correlation calculation for choices in categories C 3 and C 2, using TS2_3

From the 50 current ATCs, and based on the information from Figure 48, the following were observed and the respective new TCs added (Appendix F.9)¹⁵:

- From four ATCs exercising Ch 2.1, one exercises Ch 3.1, two exercise Ch 3.2, and one exercises Ch 3.3. No TCs were added for this case.
- From seven ATCs exercising Ch 2.2, one exercises Ch 3.1, three exercise Ch 3.2, and three exercise Ch 3.3. No TCs were added for this case.
- From seven ATCs exercising Ch 2.3, one exercises Ch 3.1, six exercise Ch 3.2. No combination to Ch 3.3 (Sundays) is feasible as it defines property [Sunday] and Ch 2.3 defines selector [if !Sunday]. Four ATCs were added (contrasting test22_ts2_3, test26_ts2_3, test3_ts2_3 and test6_ts2_3) with Ch 3.1 selected rather than Ch 3.2, resulting in EC 4: test33_ts2_3 to test36_ts2_3 respectively. The other two ATCs exercising Ch 3.2 (from the six above) could not have the output influenced by Ch 3.1 as they already exercise EC 4.
- From ten ATCs exercising Ch 2.4, one exercises Ch 3.1, nine exercises Ch 3.2. No combination to Ch 3.3 (Sundays) is feasible as it defines property [Sunday] and Ch 2.3 defines selector [if !Sunday]. Five ATCs were added (contrasting eight current TCs exercising Ch 3.2: test14 (and test23_ts2_3), test20_ts2_3, test21_ts2_3 (and test64), test24, and test3 (or test58)) with Ch 3.1 selected rather than Ch 3.2, resulting in EC 4: test37_ts2_3 to test41_ts2_3 respectively. The other one ATC exercising Ch 3.2 (from the nine above) could not have the output influenced by Ch 3.1 as it already exercises EC 4.
- From six ATCs exercising Ch 2.5, one exercises Ch 3.1, five exercise Ch 3.2. No combination to Ch 3.3 (Sundays) is feasible as it defines property [Sunday] and Ch 2.3 defines selector [if !Sunday]. Five ATCs were added with Ch 3.1 selected,

resulting in EC 4: test42_ts2_3 to test46_ts2_3 (contrasting test24_ts2_3, test25, test27_ts2_3, test4, and test7_ts2_3 respectively).

- From six ATCs exercising Ch 2.6, one exercises Ch 3.1, five exercise Ch 3.2. No combination to Ch 3.3 (Sundays) is feasible as it defines property [Sunday] and Ch 2.3 defines selector [if !Sunday]. Four ATCs were added with Ch 3.1 selected, resulting in EC 4: test47_ts2_3 to test50_ts2_3 (contrasting test16, test28_ts2_3, test31_ts2_3, and test4_ts2_3 respectively). The other one ATC exercising Ch 3.2 (from the five above) could not have the output influenced by Ch 3.1 as it already exercises EC 4.
- From five ATCs exercising Ch 2.7, one exercises Ch 3.1, three exercise Ch 3.2, and one exercises Ch 3.3. No TCs were added as Ch 2.7 (Illegal time) is an Error choice that determines the OEC EC 4 independent from any other choices selected.
- The five ATCs exercising Ch 2.8, exercise Ch 3.3. No TCs were added as Ch 2.8 defines "... && dayOfWeek.equals("Sunday")" and therefore can only be combined to choice Ch 3.3 in C 3.

Besides C 3, C 5 choices were not used in the ATS and the category more likely to be correlated¹³ to C 5 in the ATS is C 1 (Area of destination), from Figure 49 we observe a Probability of no association $P = 0.229486$.

C 5	C 1				
		Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4
	Ch 5.1	-1.291	-0.554	1.630	0.995
	Ch 5.2	1.291	0.554	-1.630	-0.995

Total test statistic (Chi Squared approximation): 4.314 with 3 degrees of freedom
Probability of no association* $P = 0.229486$

Figure 49 Correlation calculation for choices in categories C 5 and C 1, using TS2_3

From the 50 current ATCs, and based on the information from Figure 49, the following were observed and the respective new TCs added (Appendix F.9)¹⁵:

- From 18 ATCs exercising Ch 1.1, one exercises Ch 5.1, 17 exercise Ch 5.2. From the 17 ATCs exercising Ch 5.2, only 14 would have a contrasting ATC with the OEC changed to EC 4 by selecting Ch 5.1 instead of Ch 5.2. Among these 14 ATCs, only 12 different combinations are exercised, therefore 12 ATCs were added (contrasting

test11, test19_ts2_3 (or test20_ts2_3), test25_ts2_3, test26_ts2_3, test27_ts2_3, test28_ts2_3, test29_ts2_3, test3 (or test58), test3_ts2_3, test4, test4_ts2_3, and test8) with Ch 5.1 selected, resulting in EC 4: test51_ts2_3 to test62_ts2_3 respectively.

- From 19 ATCs exercising Ch 1.2, two exercise Ch 5.1, 17 exercise Ch 5.2. From the 17 ATCs exercising Ch 5.2, only 14 would have contrasting ATCs with the OEC changed to EC 4 by selecting Ch 5.1 instead of Ch 5.2. Among these 14 ATCs, only 12 different combinations are exercised, therefore 12 ATCs were added (contrasting test14 (or test23_ts2_3), test16, test17, test21_ts2_3 (or test64), test22_ts2_3, test24_ts2_3, test2_ts2_3, test30_ts2_3, test31_ts2_3, test32_ts2_3, test6_ts2_3, and test7_ts2_3) with Ch 5.1 selected, resulting in EC 4: test63_ts2_3 to test74_ts2_3 respectively.
- From ten ATCs exercising Ch 1.3, three exercise Ch 5.1, seven exercise Ch 5.2. From the seven ATCs exercising Ch 5.2, only four would have contrasting ATCs with the OEC changed to EC 4 by selecting Ch 5.1 instead of Ch 5.2. Among these four ATCs, all exercise different combinations, therefore four ATCs were added (contrasting test24, test25, test30, and test31) with Ch 5.1 selected, resulting in EC 4: test75_ts2_3 to test78_ts2_3 respectively.
- From three ATCs exercising Ch 1.4, one exercises Ch 5.1, two exercise Ch 5.2. No TCs were added as Ch 1.4 (Destination not informed or unknown) is an Error choice that determines the OEC EC 4 independent from any other choices selected.

Iteration 7

At Figure 50, 189 Nonconformities are reported and the DT has size 44 with 34 leaves. Misclassifications of two ATCs (test22_ts2_3 and test7_ts2_3) occur in two different branches of the DT, and all categories are now reported as used in the DT. From the ten Empty Leaves, nine are automatically verified as unfeasible and one is to be investigated: E3: [Ch 5.2, Ch 3.2, Ch 1.1, Ch 2.2, EC 4], which is actually unfeasible as this combination of choices could never produce OEC EC 4 (Error in input data). From the 177 Unused Combinations of Choices, 169 are verified by the tool as unfeasible combinations or feasible but already used in the ATS, and eight feasible combinations are added to the ATS as test79_ts2_3 to test86_ts2_3 (Appendix F.9).

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities Identified				
	All	Analyzed		
Misclassifications	2	0		
Unused CP Elements	0	0		
Empty Leaves	10	0		
Unused Combinations of Choices	177	0		
<i>Nonconformities Total</i>	189	0		

Figure 50 Summary of results for TS2_3, Iteration 7

Iteration 8

At Figure 51, 181 Nonconformities are reported and the DT size is 42 with 33 leaves. All CP Elements are used in the DT. All nine Empty Leaves are automatically verified as unfeasible combinations of choices, and all 168 Unused Combinations of Choices are either unfeasible or feasible but already exercised in the ATS.

Summary	Misclassifications	Unused CP Elements	Empty Leaves	Unused Combinations of Choices
Nonconformities Identified				
	All	Analyzed		
Misclassifications	4	0		
Unused CP Elements	0	0		
Empty Leaves	9	0		
Unused Combinations of Choices	168	0		
<i>Nonconformities Total</i>	181	0		

Figure 51 Summary of results for TS2_3, Iteration 8

The four misclassified ATCs occur in three different branches (Figure 52) and the suggested possible causes, Missing category or choice or Ill-defined choices, are not applicable to the case. The actual OECs for the misclassified ATCs were: EC 4 for test10_ts2_3 and test9_ts2_3, and EC 7 for test22_ts2_3 and test7_ts2_3. With no further actions to be taken, the TS with 104 TCs at Iteration 8 is the Melba result for TS2_3.

Summary: Misclassifications Unused CP Elements Empty Leaves Unused Combinations of Choices			
Leaf with misclassifications	Misclassified TCs	Possible causes	Relevant information
M1 Ch 5.2 Ch 3.2 Ch 1.3 EC 7	2 test10 ts2_3 test9 ts2_3	Investigate Missing category or choice Ill-defined choices	Investigate (C 5, Ch 5.2) also used in other branches with misclassifications: M2 M3 (C 3, Ch 3.2) also used in other branches with misclassifications: M2 M3
M2 Ch 5.2 Ch 3.2 Ch 1.2 Ch 2.3 EC 6	1 test22 ts2_3	Investigate Missing category or choice Ill-defined choices	Investigate (C 5, Ch 5.2) also used in other branches with misclassifications: M1 M3 (C 3, Ch 3.2) also used in other branches with misclassifications: M1 M3 (C 1, Ch 1.2) also used in other branches with misclassifications: M3
M3 Ch 5.2 Ch 3.2 Ch 1.2 Ch 2.5 EC 6	1 test7 ts2_3	Investigate Missing category or choice Ill-defined choices	Investigate (C 5, Ch 5.2) also used in other branches with misclassifications: M1 M2 (C 3, Ch 3.2) also used in other branches with misclassifications: M1 M2 (C 1, Ch 1.2) also used in other branches with misclassifications: M2

Figure 52 Misclassified ATCs for TS2_3 at Iteration 8

5.3.3. Summarizing Resulting Test Suites

The final test suites resulting from this process are the TS2_2Melba, composed of 58 TCs, and the TS2_3Melba, composed of 103 TCs, which are respectively Melba improvements of TS2_2 and TS2_3. Table 10 outlines the comparison of the code coverage and mutation score on the Taxi Billing system for these test suites against the original test suites.

Test Suite: size	Line Coverage 159 lines	Branch Coverage 122 branches	Traditional Mutants Generated: 308		Class-level Mutants Generated: 69	
	% (lines)	% (branches)	Killed	Total Score %	Killed	Total Score %
TS2 1: 73	100 (159)	89 (109)	232	75.32	23	33.33
TS2 2: 31	88 (141)	71 (87)	199	64.61	23	33.33
TS2 2Melba: 58	97 (155)	88 (108)	230	74.67	23	33.33
TS2 3: 18	83 (133)	62 (76)	182	59.09	23	33.33
TS2 3Melba: 104	98 (157)	85 (104)	229	74.35	23	33.33

Table 10 Comparison of performance of resulting TS and original ones for Experiment 2

From the summary of the resulting TS of Experiment 1 (Section 5.2.4) we already know the branches that were not covered by the original test suite TS2_1, and the mutants that it could not detect in the code. For the test suites TS2_2Melba and TS2_3Melba we know that the class-level mutation scores were the same as the original one and the

mutants alive were verified to be the same as well. Next, we analyze the numbers that were different from the original one and identify the test cases that could have completed each of the test suites.

For the test suite TS2_2Melba:

- Four uncovered lines indicating missing TCs for the following three cases: Monday to Saturday with departureTime between 0 (inclusive) and 7 and suburban destination; Monday to Saturday with departureTime between 17 (inclusive) and 19 and suburban destination; Monday to Saturday with departureTime between 19 (inclusive) and 24 and beyond suburban destination.
- 14 uncovered branches indicating missing TCs for the following cases: Same three reasons as above; Missing combinations of departureTime intervals for Sunday or Monday to Saturday, with the error situation of not informed destinationArea. This are however not really missing TCs as testing an error situation once is part of our heuristics (Section 3.4.1.3); Uncovered parts of conditions in the same statement (e.g.: the statement *if (departureTime >= 19 && departureTime < 24)* where only one condition was evaluated to both true and false) again not representing missing TCs as the statement was already evaluated to both true and false.
- The 78 Live Traditional Mutants where: The 67 that are in common with mutants alive for TS2_1; Eleven mutants killed by TS2_1 are left alive: AOIS_17, AOIS_19, AOIS_21 (additions of operator ++ at *departureTime*), COI_31, COI_37, COI_62 (use of operator not - ! - for *destinationArea* or *DayOfWeek*), LOI_19 (use of operator ~ for *departureTime*), ROR_32, ROR_39, ROR_60, ROR_72 (change of comparison operator, for instance == instead of !=); It is interesting to note that nine mutants, not killed by TS2_1, were killed by TS2_2Melba: AOIS_55, AOIS_57, AOIS_59, AOIS_61, ROR_14, ROR_35, ROR_36, ROR_67, ROR_68.

For the test suite TS2_3Melba:

- Two uncovered lines indicating missing TC for the following case: Monday to Saturday with departureTime between 0 (inclusive) and 7 and beyond suburban destination.

- 18 uncovered branches indicating missing TCs for the following cases: Same reason as the one above; Missing combinations of departureTime intervals for Sunday or Monday to Saturday, with the error situation of not informed destinationArea. However, these are not really missing TCs as testing an error situation once is part of our heuristics (Section 3.4.1.3); Uncovered parts of conditions in the same statement, similar to the cases of TS2_2Melba with some additional statements.
- The 79 Live Traditional Mutants where: The 64 that are in common with mutants alive for TS2_1; 15 killed by TS2_1 are left alive: AOIS_17, AOIS_19, AOIS_21 (additions of operator ++ at departureTime), COI_22, COI_31, COI_38 (use of operator not - ! - for destinationArea or DayOfWeek), COR_3 (change of logical operator && for || in departureTime condition), LOI_7 (use of operator ~ for departureTime), ROR_27, ROR_32, ROR_33, ROR_37, ROR_40, ROR_41, ROR_45 (change of comparison operator, for instance > instead of >=); It is interesting to note that 12 mutants, not killed by TS2_1, were killed by TS2_3Melba: AOIS_11, AOIS_13, AOIS_3, AOIS_5, AOIS_7, AOIS_9, ROR_14, ROR_35, ROR_36, ROR_6, ROR_67, ROR_68.

Apart from completeness and effectiveness, we wanted to evaluate the quality of the resulting test suites by counting the repeated combinations of choices and OEC, i.e. redundant ATCs, in each of them in comparison to the same count in TS2_1. Table 11 shows that the number of redundant ATCs has actually decreased in the two resulting test suites: the original TS2_1 has ten repeated ATCs (one for each of ten redundant combinations); TS2_2Melba has five repeated ATCs (in four redundant combinations); and TS2_3Melba has seven repeated ATCs (one for each of seven redundant combinations).

Test Suite: size	Combinations	TCs	TCs Names
TS2_1: 73	1. [Ch 1.2, Ch 2.3, Ch 3.2, Ch 4.2, Ch 5.2, EC 6]	2	test13, test66
	2. [Ch 1.2, Ch 2.4, Ch 3.2, Ch 4.2, Ch 5.2, EC 6]	2	test14, test67
	3. [Ch 1.2, Ch 2.5, Ch 3.2, Ch 4.2, Ch 5.2, EC 6]	2	test15, test68
	4. [Ch 1.1, Ch 2.3, Ch 3.2, Ch 4.2, Ch 5.2, EC 6]	2	test2, test7
	5. [Ch 1.3, Ch 2.2, Ch 3.2, Ch 4.2, Ch 5.2, EC 7]	2	test22, test27
	6. [Ch 1.1, Ch 2.4, Ch 3.2, Ch 4.2, Ch 5.2, EC 5]	2	test3, test58
	7. [Ch 1.1, Ch 2.4, Ch 3.2, Ch 4.1, Ch 5.2, EC 6]	2	test57, test59
	8. [Ch 1.2, Ch 2.3, Ch 3.2, Ch 4.1, Ch 5.2, EC 7]	2	test60, test63
	9. [Ch 1.2, Ch 2.4, Ch 3.2, Ch 4.1, Ch 5.2, EC 7]	2	test61, test64
	10. [Ch 1.2, Ch 2.5, Ch 3.2, Ch 4.1, Ch 5.2, EC 7]	2	test62, test65
TS2_2 Melba: 58	1. [Ch 1.2, Ch 2.2, Ch 3.1, Ch 4.2, Ch 5.1, EC 4]	2	test10_Scenario2, test11_Scenario2
	2. [Ch 1.1, Ch 2.4, Ch 3.2, Ch 4.1, Ch 5.2, EC 6]	2	test19_Scenario2, test21_Scenario2
	3. [Ch 1.1, Ch 2.4, Ch 3.2, Ch 4.2, Ch 5.2, EC 5]	3	test20_Scenario2, test3, test58
	4. [Ch 1.2, Ch 2.4, Ch 3.2, Ch 4.1, Ch 5.2, EC 7]	2	test61, test64
TS2_3 Melba: 104	1. [Ch 1.2, Ch 2.4, Ch 3.2, Ch 4.2, Ch 5.2, EC 6]	2	test14, test23 ts2 3
	2. [Ch 1.1, Ch 2.4, Ch 3.1, Ch 4.1, Ch 5.2, EC 4]	2	test14 ts2 3, test38 ts2 3
	3. [Ch 1.1, Ch 2.4, Ch 3.2, Ch 4.1, Ch 5.2, EC 6]	2	test19 ts2 3, test20 ts2 3
	4. [Ch 1.2, Ch 2.4, Ch 3.2, Ch 4.1, Ch 5.2, EC 7]	2	test21 ts2 3, test64
	5. [Ch 1.1, Ch 2.4, Ch 3.2, Ch 4.2, Ch 5.2, EC 5]	2	test3, test58
	6. [Ch 1.2, Ch 2.2, Ch 3.2, Ch 4.1, Ch 5.1, EC 4]	2	test50, test65 ts2 3
	7. [Ch 1.2, Ch 2.2, Ch 3.3, Ch 4.2, Ch 5.2, EC 7]	2	test79 ts2 3, test86 ts2 3

Table 11 Combinations of choices and OEC exercised by more than 1 ATC in the same ATS

This experiment has shown that:

- The Melba method has definitely helped improve both TS2_2 and TS2_3, both in terms of coverage and effectiveness. Future work should continue this sort of study, with various initial coverage percentages, in the attempt to identify a minimum percentage of code coverage for the initial test suite above which Melba shows to be useful but under which Melba may not help;
- The coverage of code of TS2_2Melba reached a very close percentage to the original one with fewer test cases. This was not the case for TS2_3Melba that increased the number of TCs by more than 42% and yet could not reach the same code coverage of TS2_1. This may be due to our heuristic for adding test cases. Other heuristics should be investigated in future work;

- Both of the abstract test suites resulting from Melba present less redundancy of ATCs than the original ATS. This could explain some of the mutants that were killed by TS2_1 and were not killed by TS2_2Melba or TS2_3Melba, or some of the statements covered by TS2_1 that were not covered by TS2_2Melba or TS2_3Melba. Recall that despite the fact that the ATCs may be the same for two different concrete test cases, the concrete test cases may exercise (cover) distinct parts of the code and kill slightly different mutants because they may have different input values. This is an important factor that should be looked at in future investigations when dealing with the elimination of test cases for Case 4: Too Many Test Cases for a Rule;
- The better variability of ATCs permitted to reach a very close effectiveness to the original one in terms of detecting mutants, however some mutants previously detected were left alive and others were detected instead. For both the additional mutants killed and code covered by the improved test suites, which were respectively not killed and not covered by TS2_1, this may have been a result of our heuristic and sequence of steps adopted in the experiment. Future investigations should look into how different approaches on how to solve the nonconformities really affect the final results, e.g.: in which order to treat the nonconformities, how many test cases to add per iteration;
- In order to analyze and apply corrections to the data, a high level of understanding and visual analysis of the DT, together with the domain of the system specification, are required from the tester;
- Last, with respect to the Melba process, we found that it was difficult to address misclassifications upfront. Instead, we systematically tried to address simpler problems such as missing choices (combinations). We feel that the Melba process could be improved in that regard, either by providing additional results to help in the analysis of misclassifications or simply by suggesting an order to solve nonconformities. In our experiments we have tried that latter solution, by trying to address unused elements and combinations first, and analyzing misclassifications last, in case they were still reported.

5.4. Experiment 3

In this experiment we want to evaluate question III: *What is the impact the selection of choices has on the method when adding TCs?* We start from a TS that is not complete for the system under test and definitely needs to be augmented, since we want to improve it and see the impact of choices selection on the result of the improvement. Then, we try different choice selection heuristics when adding test cases and we analyze their impact on the subsequent versions of the DT, and consequently on the results of the Melba process. Our approach is to observe when a choice or an incomplete combination of choices¹⁷ is verified to be missing in the ATS, and then to analyze which other choices (from other categories) can be selected to create test cases, and what is the impact of the selection.

In Subsection 5.4.1 we describe the experiment design, including the CP Specification used in this experiment, and the initial test suite to be improved. In Subsection 5.4.2 we describe the results obtained when applying the Melba process, and in Subsection 5.4.3 the results are summarized.

5.4.1. Experiment Design

In order to apply distinct heuristics for the selection of choices for the new test cases, we first define the problem. Suppose n categories $C_1, C_2, \dots, C_n$, where each category has m choices (this is a simplification that does not impact the generality of the discussion): $Ch_{1.1}, \dots, Ch_{1.m}, Ch_{2.1}, \dots, Ch_{2.m}, \dots, Ch_{n.1}, \dots, Ch_{n.m}$. When creating a test case, one has to select a choice for each of the n categories, and select a value for that choice. In general when Melba suggests that a feasible combination of choices is missing, that combination does not involve all the categories (i.e., a choice for each category). For instance Melba may report that combination $Ch_{1.1}, Ch_{2.1}, Ch_{3.1}, Ch_{n.4}$ is missing, where categories 4 to $n-1$ are not mentioned.

When adding new test cases for missing combinations of choices, we therefore want to answer to the following questions:

¹⁷ An incomplete combination of choices is defined here as an unused choice or a missing combination of at least two choices, and it will be referred to as a (feasible missing) combination of choices.

1. How to select choices for categories not mentioned in the missing combination. In our example, how to select choices for categories $C_4 \dots C_{n-1}$?
2. Does the selection heuristic matter in terms of the resulting DT, e.g., in terms of the number of rules in the DT, number of misclassifications, and number of CP element not used in the DT?
3. Does the selection heuristic matter in terms of code coverage and fault detection of the resulting test suite?

To answer these three questions, we will use the two choice selection heuristics described in Section 5.4.1.1.

In summary, the original TS will be augmented with new TCs up to the point where the method does not suggest that any combination of choices, or unused choice as in our definitions for this experiment, is missing in the ATS. This augmentation will occur in two distinct ways: first via heuristic 1 and second via heuristic 2 (Section 5.4.1.1). Since we are interested in the impact of our heuristics on the Melba process when adding test cases for reported missing combinations and for unused CP elements (i.e., choices not used in the ATS), i.e., for solving only two types of nonconformities, we will only solve occurrences of these nonconformities during our experiments. We do not want to solve misclassifications or strongly associated CP elements (used in the ATS but not used in the DT), as these improvements might hide the impact of our heuristics. Therefore, during our experiments, once no feasible combinations are indicated as missing in the ATS, we stop the Melba process (even if other nonconformities could be addressed) and results will be ready for evaluation of heuristics impact.

5.4.1.1. Heuristics to select choices for new Test Cases

When Melba reports a missing combination of two or more choices, thereby explicitly indicating a missing combination of choices and omits choices for other categories, our strategy is to first determine how many times, in the ATS, each choice of the omitted categories is combined with each choice of the reported missing combination.

Reusing our example of n categories above, we first identify how many times in the ATS:

- Ch 4.1 is combined with each of Ch 1.1, Ch 2.1, Ch 3.1, and Ch $n.4$, referred to as `count(Ch 4.1)`;

- Ch 4.2 is combined with each of Ch 1.1, Ch 2.1, Ch 3.1, and Ch n.4, referred to as count(Ch 4.2);
- ...
- Ch n-1.m is combined with each of Ch 1.1, Ch 2.1, Ch 3.1, and Ch n.4, referred to as count(Ch n-1.m).

Additionally, when Melba reports a missing choice Ch i.j of category C i, and omits choices for other categories, our strategy is to first determine how many times, in the ATS, choices in categories other than C i, say Ch k.l, are combined with the choices in C i, referred to as count(Ch k.l).

Given that information, we define two heuristics for the selection of the choices to complete a missing combination:

Heuristic 1 – For each missing category, select the choice with the minimum count.

Heuristic 2 – For each missing category, select the choice with the maximum count.

The rationale for heuristic 1 is to obtain test cases with high variability in terms of combinations of choices, whereas the rationale for heuristic 2 is to obtain new test cases that do not differ much from existing ones (in terms of combinations of choices) in the hope that the impact of the missing combination (which is part of the new test case) on the learning process will be important (e.g., better decision tree, improved Melba process).

In each heuristic, if more than one choice has a minimum/maximum count value, any one of them can be selected. Also, if combining the choices of the missing combination with the choice that has the minimum/maximum count value is impossible (unfeasible combination of choices as per choice constraints), then we select the next smallest/greatest count value. If no choice of can be feasibly combined, then we set a N/A value for the category.

In our above example with n categories, heuristic 1 selects the choice Ch i.j such that $Ch\ i.j = \text{Min}_{(k,l) \in \{(4,1), (4,2), \dots, (n-1,m)\}} \text{count}(Ch\ k.l)$, and heuristic 2 selects the choice Ch i.j such that $Ch\ i.j = \text{Max}_{(k,l) \in \{(4,1), (4,2), \dots, (n-1,m)\}} \text{count}(Ch\ k.l)$.

It is important to remember that the DT is built based on the ATS, therefore only the ATCs have influence on it. For that reason, it is obvious that adding test cases for heuristics 1 and 2 with the same combinations of choices but different parameters' values

(i.e., values derived from choices' definitions) would lead to identical (DT) results, as the ATCs for those TCs would be the same. Using varying input values may only impact structural coverage and mutation score, and studying different strategies for selecting test case inputs (from choices' definitions) is left for future work.

5.4.1.2. The CP Specification

As we are interested in evaluating and improving a test suite, the ideal CP Specification to be used in this experiment is one considered as most complete and accurate as possible. Hence, we use the one resulting from Experiment 1, found in Appendix F.3.

5.4.1.3. The Test Suite

The purpose is to improve a TS by adding new TCs according to the suggestions of Melba and trying different possibilities. Since we need a test suite we can improve on, we selected the smallest TS defined in Experiment 2: TS2_3 (Appendix F.10), with 18 Test Cases, covering 83% lines, and 62% branches of the Taxi Billing code.

5.4.2. Applying Melba method

Iteration 1

The total number of Nonconformities for TS2_3 is 26 and the DT is:

```
J48 unpruned tree
-----
C 5 = Ch 5.2
|   C 1 = Ch 1.1
|   |   C 3 = Ch 3.3: EC 6 (2.0/1.0) M1
|   |   C 3 = Ch 3.2: EC 5 (4.0/2.0) M2
|   C 1 = Ch 1.2: EC 7 (4.0/1.0) M3
|   C 1 = Ch 1.3: EC 7 (4.0)
|   C 1 = Ch 1.4: EC 4 (1.0)
C 5 = Ch 5.1: EC 4 (3.0)

Number of Leaves   :    6

Size of the tree   :    9
```

There are four Misclassifications in three different branches, eleven Unused CP Elements (choices in categories C 2, C 4, and choice Ch 3.1), no Empty Leaf, and eleven Unused Combinations of Choices. In the eleven unused CP elements, choices Ch 2.1 and Ch 3.1 were verified by the tool as not being exercised in the ATS. By adding TCs for the unused combinations, we want to observe whether these unused choices will be included

in the ATS in further iterations. From the eleven unused combinations of choices, only one is suggested as feasible and not used in the ATS: [(C 5,Ch 5.2) (C 1,Ch 1.2) (C 3,Ch 3.3)]. For the new TC to be added, we need to select choices for categories C 2 and C 4. From the ATS, we count the number of times choices in these two categories are used with the suggested choices in the combination: C 2 (Table 12) and C 4 (Table 13).

Choices	Ch 2.1	Ch 2.2	Ch 2.3	Ch 2.4	Ch 2.5	Ch 2.6	Ch 2.7	Ch 2.8
Ch 5.2	0	3	1	5	2	1	1	2
Ch 1.2	0	2	0	2	0	1	0	0
Ch 3.3	0	2	0	0	0	0	0	2
Total	0	7	1	7	2	2	1	4

Table 12 Choices in C 2 combined with choices in the first missing combination

Choices	Ch 4.1	Ch 4.2
Ch 5.2	6	9
Ch 1.2	3	2
Ch 3.3	2	2
Total	11	13

Table 13 Choices in C 4 combined with choices in the first missing combination

5.4.2.1. Heuristic 1

Following heuristic 1, we select minimum counts from Table 12 and Table 13. We obtain combination [(C 5,Ch 5.2) (C 1,Ch 1.2) (C 3,Ch 3.3) (C 2,Ch 2.1) (C 4,Ch 4.1)]. TC test1_option1 is added (all new TCs added are listed at Appendix F.11).

Iteration 2

The DT obtained is now:

```

J48 unpruned tree
-----
C 5 = Ch 5.2
| C 2 = Ch 2.8: EC 6 (2.0/1.0) M1
| C 2 = Ch 2.4
| | C 1 = Ch 1.2: EC 6 (2.0/1.0) M2
| | C 1 = Ch 1.3: EC 7 (1.0)
| | C 1 = Ch 1.1: EC 5 (2.0)
| | C 1 = Ch 1.4: EC 6 (0.0) E1
| C 2 = Ch 2.6: EC 7 (1.0)
| C 2 = Ch 2.2: EC 7 (3.0)
| C 2 = Ch 2.1: EC 4 (1.0)
| C 2 = Ch 2.5: EC 6 (2.0/1.0) M3
| C 2 = Ch 2.3: EC 4 (1.0)
| C 2 = Ch 2.7: EC 4 (1.0)
C 5 = Ch 5.1: EC 4 (3.0)

Number of Leaves :    12

Size of the tree :    15

```

The number of Nonconformities reported is 48 with: three Misclassifications; five Unused CP Elements (choices in categories C 3 and C 4); one Empty Leaf (unfeasible combination of choices and OEC); and 39 Unused Combinations of Choices, where seven are feasible combinations not used in the ATS. From the five unused CP elements, only Ch 3.1 was verified by the tool as not being exercised in the ATS. By adding TCs for the unused combinations, we want to observe whether this choice will be included in the ATS in further iterations. For the seven missing combinations we need to select choices for categories C 3 and C 4. These combinations as well as the number of times choices in C 3 and C 4 are used with the suggested choices in the combinations are:

1. [(C 5,Ch 5.2) (C 2,Ch 2.8) (C 1,Ch 1.2)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	0	11	5	-	7	9
Ch 2.8	0	0	2	-	2	0
Ch 1.2	0	5	1	-	4	2
Total	0	16	8	-	13	11

Heuristic 1 leads to the selection of Ch 3.1 and Ch 4.2, but Ch 3.1 can't be combined to Ch 2.8 (Sundays special time of departure), then it must be combined to Ch 3.3, that has the next smallest count value. Therefore, Ch 3.3 and Ch 4.2 are selected for test2_option1.

2. [(C 5,Ch 5.2) (C 2,Ch 2.6) (C 1,Ch 1.3)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	0	11	5	-	7	9
Ch 2.6	0	2	0	-	0	2
Ch 1.3	0	4	2	-	2	4
Total	0	17	7	-	9	15

Heuristic 1 leads to the selection of Ch 3.1 and Ch 4.1 for test3_option1.

3. [(C 5,Ch 5.2) (C 2,Ch 2.6) (C 1,Ch 1.1)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	0	11	5	-	7	9
Ch 2.6	0	2	0	-	0	2
Ch 1.1	0	4	2	-	2	4
Total	0	17	7	-	9	15

Heuristic 1 leads to the selection of Ch 3.1 and Ch 4.1 for test4_option1.

4. [(C 5,Ch 5.2) (C 2,Ch 2.5) (C 1,Ch 1.2)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	0	11	5	-	7	9
Ch 2.5	0	2	0	-	0	2
Ch 1.2	0	5	1	-	4	2

Total	0	18	6	-	11	13
-------	---	----	---	---	----	----

Heuristic 1 leads to the selection of Ch 3.1 and Ch 4.1 for test5_option1.

5. [(C 5,Ch 5.2) (C 2,Ch 2.3) (C 1,Ch 1.2)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	0	11	5	-	7	9
Ch 2.3	0	2	0	-	2	0
Ch 1.2	0	5	1	-	4	2
Total	0	18	6	-	13	11

Heuristic 1 leads to the selection of Ch 3.1 and Ch 4.2 for test6_option1.

6. [(C 5,Ch 5.2) (C 2,Ch 2.3) (C 1,Ch 1.3)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	0	11	5	-	7	9
Ch 2.3	0	2	0	-	2	0
Ch 1.3	0	4	2	-	2	4
Total	0	17	7	-	11	13

Heuristic 1 leads to the selection of Ch 3.1 and Ch 4.1 for test7_option1.

7. [(C 5,Ch 5.2) (C 2,Ch 2.3) (C 1,Ch 1.1)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	0	11	5	-	7	9
Ch 2.3	0	2	0	-	2	0
Ch 1.1	0	4	2	-	2	4
Total	0	17	7	-	11	13

Heuristic 1 leads to the selection of Ch 3.1 and Ch 4.1 for test8_option1.

Iteration 3

The DT obtained is now:

J48 unpruned tree

```

C 2 = Ch 2.8: EC 7 (3.0/1.0) M1
C 2 = Ch 2.4
| C 1 = Ch 1.2: EC 6 (2.0/1.0) M2
| C 1 = Ch 1.3: EC 7 (1.0)
| C 1 = Ch 1.1: EC 5 (2.0)
| C 1 = Ch 1.4: EC 6 (0.0) E1
C 2 = Ch 2.6: EC 4 (4.0/1.0) M3
C 2 = Ch 2.2: EC 7 (4.0/1.0) M4
C 2 = Ch 2.1: EC 4 (1.0)
C 2 = Ch 2.5: EC 6 (3.0/2.0) M5
C 2 = Ch 2.3: EC 4 (5.0)
C 2 = Ch 2.7: EC 4 (1.0)

```

Number of Leaves : 11

Size of the tree : 13

The number of Nonconformities reported is 42, with: six Misclassifications in five different branches; seven Unused CP Elements (choices in categories C 3, C 4, and C 5 – all in the ATS – where C 3 was found to be strongly associated to C 2); one Empty Leaf

(unfeasible combination of choices and OEC); and 28 Unused Combinations of Choices, all unfeasible combinations or feasible combinations already used in the ATS. As all choices defined in the CP Specification are now automatically verified to be exercised in the ATS and no combination of choices is verified to be missing in the ATS, results of Heuristic 1 at Iteration 3 will be evaluated (Section 5.4.3).

5.4.2.2.Heuristic 2

Following heuristic 2, we select maximum counts from Table 12 and Table 13. We randomly choose between Ch 2.2 and Ch 2.4 for C 2 as they have the same count, and select Ch 4.2 for C 4, resulting in the complete combination [(C 5,Ch 5.2) (C 1,Ch 1.2) (C 3,Ch 3.3) (C 2,Ch 2.2) (C 4,Ch 4.2)]. TC test1_option2 is added to the TS (all new TCs added are listed at Appendix F.11).

Iteration 2

The DT obtained is now:

```
J48 unpruned tree
-----
C 5 = Ch 5.2
|   C 1 = Ch 1.1
|   |   C 3 = Ch 3.3: EC 6 (2.0/1.0) M1
|   |   C 3 = Ch 3.2: EC 5 (4.0/2.0) M2
|   |   C 1 = Ch 1.2: EC 7 (5.0/1.0) M3
|   |   C 1 = Ch 1.3: EC 7 (4.0)
|   |   C 1 = Ch 1.4: EC 4 (1.0)
|   C 5 = Ch 5.1: EC 4 (3.0)

Number of Leaves   :    6
Size of the tree   :    9
```

The number of Nonconformities reported is 26, with: four Misclassifications in three different branches, eleven Unused CP Elements (choices in categories C 2 and C4, and choice Ch 3.1), no Empty Leaf, and eleven Unused Combinations of Choices, all unfeasible combinations or feasible combinations already used in the ATS. From the unused CP elements, choices Ch 2.1 and Ch 3.1 were verified by the tool as not being exercised in the ATS. To define new TC(s) exercising them, as outlined in Section 5.4.1.1, the combination of choices in categories other than C 2 and C 3 will be determined by the number of times choices in C 1, C 4, and C 5 were combined with each of the other choices in C 2 (Table 14), and the same for the choices in C 3 (Table 15).

Choices	Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4	Ch 4.1	Ch 4.2	Ch 5.1	Ch 5.2
Ch 2.2	1	3	1	0	2	3	1	4
Ch 2.3	0	0	1	1	2	0	1	1
Ch 2.4	2	2	1	0	1	4	0	5
Ch 2.5	1	0	1	0	0	2	0	2
Ch 2.6	0	1	1	0	0	2	1	1
Ch 2.7	1	0	0	0	1	0	0	1
Ch 2.8	1	0	1	0	2	0	0	2
Total	6	6	6	1	8	11	3	16

Table 14 Choices in C 2, other than Ch 2.1, combined with choices in C 1, C 4, C 5

Choices	Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4	Ch 4.1	Ch 4.2	Ch 5.1	Ch 5.2
Ch 3.2	4	5	4	1	6	8	3	11
Ch 3.3	2	1	2	0	2	3	0	5
Total	6	6	6	1	8	11	3	16

Table 15 Choices in C 3, other than Ch 3.1, combined with choices in C 1, C 4, C 5

For both cases, the selection of choices will be the same: for category C 1: the choice with maximum count value is any of Ch 1.1, Ch 1.2 or Ch 1.3, and Ch 1.2 was randomly selected; for C 4, Ch 4.2 has the maximum count value; and for C 5, Ch 5.2 is the one selected. Therefore the new ATC will exercise combination: [(C 1,Ch 1.2) (C 2,Ch 2.1) (C 3,Ch 3.1) (C 4,Ch 4.2) (C 5,Ch 5.2)], added to the ATS as test2_option2.

Iteration 3

The DT obtained is now:

```

J48 unpruned tree
-----
C 5 = Ch 5.2
| C 2 = Ch 2.8: EC 6 (2.0/1.0) M1
| C 2 = Ch 2.4
| | C 1 = Ch 1.2: EC 6 (2.0/1.0) M2
| | C 1 = Ch 1.3: EC 7 (1.0)
| | C 1 = Ch 1.1: EC 5 (2.0)
| | C 1 = Ch 1.4: EC 6 (0.0) E1
| C 2 = Ch 2.6: EC 7 (1.0)
| C 2 = Ch 2.2: EC 7 (4.0)
| C 2 = Ch 2.5: EC 6 (2.0/1.0) M3
| C 2 = Ch 2.1: EC 4 (1.0)
| C 2 = Ch 2.3: EC 4 (1.0)
| C 2 = Ch 2.7: EC 4 (1.0)
C 5 = Ch 5.1: EC 4 (3.0)

Number of Leaves :    12

Size of the tree :    15

```

The number of Nonconformities reported is 48, from which three Misclassifications in three different branches, five Unused CP Elements (choices in categories C 3 and C4 – all in the ATS), one Empty Leaf (unfeasible combination of choices), and 39 Unused

Combinations of Choices, from which seven are feasible combinations missing in the ATS. It is interesting to note at this point that the DT and results are very similar to heuristic 1, Iteration 2, and so the seven suggested combinations are exactly the same, and for them we also need to select choices for categories C 3 and C 4. The count of choices used in the ATS may be different than when using heuristic 1 though, as the two abstract test suites are slightly different at this point. The combinations, as well as the number of times choices in C 3 and C 4 are used with the suggested choices in the combinations, are:

1. [(C 5,Ch 5.2) (C 2,Ch 2.8) (C 1,Ch 1.2)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	1	11	5	-	6	11
Ch 2.8	0	0	2	-	2	0
Ch 1.2	1	5	1	-	3	4
Total	2	16	8	-	11	15

Heuristic 2 leads to the selection of Ch 3.2 and Ch 4.2, but Ch 3.2 can't be combined to Ch 2.8 (Sundays special time of departure). Then we select Ch 3.3. Therefore, Ch 3.3 and Ch 4.2 are selected for test3_option2.

2. [(C 5,Ch 5.2) (C 2,Ch 2.6) (C 1,Ch 1.3)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	1	11	5	-	6	11
Ch 2.6	0	2	0	-	0	2
Ch 1.3	0	4	2	-	2	4
Total	1	17	7	-	8	17

Heuristic 2 leads to the selection of Ch 3.2 and Ch 4.2 for test4_option2.

3. [(C 5,Ch 5.2) (C 2,Ch 2.6) (C 1,Ch 1.1)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	1	11	5	-	6	11
Ch 2.6	0	2	0	-	0	2
Ch 1.1	0	4	2	-	2	4
Total	1	17	7	-	8	17

Heuristic 2 leads to the selection of Ch 3.2 and Ch 4.2 for test5_option2.

4. [(C 5,Ch 5.2) (C 2,Ch 2.5) (C 1,Ch 1.2)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	1	11	5	-	6	11
Ch 2.5	0	2	0	-	0	2
Ch 1.2	1	5	1	-	3	4
Total	2	18	6	-	9	17

Heuristic 2 leads to the selection of Ch 3.2 and Ch 4.2 for test6_option2.

5. [(C 5,Ch 5.2) (C 2,Ch 2.3) (C 1,Ch 1.2)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	1	11	5	-	6	11
Ch 2.3	0	2	0	-	2	0
Ch 1.2	1	5	1	-	3	4
Total	2	18	6	-	11	15

Heuristic 2 leads to the selection of Ch 3.2 and Ch 4.2 for test7_option2.

6. [(C 5,Ch 5.2) (C 2,Ch 2.3) (C 1,Ch 1.3)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	1	11	5	-	6	11
Ch 2.3	0	2	0	-	2	0
Ch 1.3	0	4	2	-	2	4
Total	1	17	7	-	10	15

Heuristic 2 leads to the selection of Ch 3.2 and Ch 4.2 for test8_option2.

7. [(C 5,Ch 5.2) (C 2,Ch 2.3) (C 1,Ch 1.1)]

Choices	Ch 3.1	Ch 3.2	Ch 3.3	-	Ch 4.1	Ch 4.2
Ch 5.2	1	11	5	-	6	11
Ch 2.3	0	2	0	-	2	0
Ch 1.1	0	4	2	-	2	4
Total	1	17	7	-	10	15

Heuristic 2 leads to the selection of Ch 3.2 and Ch 4.2 for test9_option2.

Iteration 4

The DT obtained is now:

J48 unpruned tree

```

-----
C 1 = Ch 1.1: EC 6 (8.0/4.0) M1
C 1 = Ch 1.2
| C 2 = Ch 2.4: EC 6 (2.0/1.0) M2
| C 2 = Ch 2.6: EC 7 (1.0)
| C 2 = Ch 2.2: EC 7 (3.0/1.0) M3
| C 2 = Ch 2.1: EC 4 (1.0)
| C 2 = Ch 2.8: EC 7 (1.0)
| C 2 = Ch 2.5: EC 6 (1.0)
| C 2 = Ch 2.3: EC 6 (1.0)
| C 2 = Ch 2.7: EC 6 (0.0) E1
C 1 = Ch 1.3
| C 5 = Ch 5.2: EC 7 (6.0)
| C 5 = Ch 5.1: EC 4 (2.0)
C 1 = Ch 1.4: EC 4 (1.0)

```

Number of Leaves : 12

Size of the tree : 15

The number of Nonconformities reported is 43, from which six Misclassifications in three different branches, six Unused CP Elements (choices in categories C 3 and C4, and the OEC EC 5 – all used in the ATS but not in the tree), one Empty Leaf (unfeasible combination of choices), and 30 Unused Combinations of Choices, all unfeasible

combinations or feasible but already used in the ATS. As all choices defined in the CP Specification are now verified by the tool to be exercised in the ATS and no combination of choices is verified to be missing in the ATS, we stop the Melba process and results of heuristic 2 at Iteration 4 will be evaluated (Section 5.4.3).

5.4.3. Summarizing Results

TS2_3Option1, composed of 26 TCs, is the resulting TS from heuristic 1, and it results from three iterations of the Melba process. TS2_3Option2, composed of 27 TCs, is the resulting TS from heuristic 2, within four iterations of the process. The summary of code coverage and mutation scores for these two test suites in comparison to the original one (TS2_3) is presented in Table 16 where it can be observed that both test suites had the effectiveness improved in comparison to the original one (except for the equivalent class-level mutants discussed in the results of Experiment 1 at Section 5.2.4), and yet TS2_3Option2 presented better results than TS2_3Option1, in both coverage percentages and mutation scores.

Test Suite: size	Line Coverage 159 lines	Branch Coverage 122 branches	Traditional Mutants Generated: 308		Class-level Mutants Generated: 69	
	% (lines)	% (branches)	Killed	Total Score %	Killed	Total Score %
TS2_3: 18	83 (133)	62 (76)	182	59.09	23	33.33
TS2_3Option1: 26	86 (137)	66 (81)	188	61.03	23	33.33
TS2_3Option2: 27	93 (149)	77 (94)	211	68.50	23	33.33

Table 16 Summary of Code Coverage and Mutation Scores for Experiment 3

The uncovered lines and branches of TS2_3Option1 reveal the following 14 missing scenarios in the test cases:

1. destinationArea = "Peripheral road", "Roissy", "Du Parc"
2. departureTime from 0 to 7 (not inclusive), dayOfWeek = Sunday, destinationArea in Suburban area¹⁸

¹⁸ Recall from the system specifications that Suburban area includes "End of area", "Orly", "Roissy", "Du Parc", and "Villepinte".

3. departureTime from 0 to 7 (not inclusive), dayOfWeek = Sunday, not informed destinationArea (error situation)
4. departureTime from 7 to 24 (not inclusive), dayOfWeek = Sunday, not informed destinationArea (error situation)
5. holiday, departureTime from 0 to 24 (not inclusive), destinationArea in Urban area¹⁹
6. holiday, departureTime from 0 to 24 (not inclusive), destinationArea Beyond Suburban
7. departureTime from 0 to 7 (not inclusive), dayOfWeek from Monday to Saturday, for all 3 groups of destination area
8. departureTime from 7 to 10 (not inclusive), dayOfWeek from Monday to Saturday, for all 3 groups of destination area
9. departureTime from 10 to 17 (not inclusive), dayOfWeek from Monday to Saturday, not informed destinationArea (error situation)
10. departureTime from 17 to 19 (not inclusive), dayOfWeek from Monday to Saturday, destinationArea in Suburban area
11. departureTime from 17 to 19 (not inclusive), dayOfWeek from Monday to Saturday, not informed destinationArea (error situation)
12. departureTime from 19 to 24 (not inclusive), dayOfWeek from Monday to Saturday, destinationArea in Urban area
13. departureTime from 19 to 24 (not inclusive), dayOfWeek from Monday to Saturday, destinationArea Beyond Suburban
14. departureTime from 19 to 24 (not inclusive), dayOfWeek from Monday to Saturday, not informed destinationArea (error situation)

The uncovered lines and branches of TS2_3Option2 reveal eleven missing scenarios, from which nine are in common with heuristic 1: cases 1 (excluding "Peripheral road"), 3, 4, 5, 6, 7, 9, 11, 14; and the additional cases:

1. departureTime from 7 to 10 (not inclusive), dayOfWeek from Monday to Saturday, not informed destinationArea (error situation)
2. invalid departureTime (less than 0 or greater than 23) and dayOfWeek = Sunday

¹⁹ Recall from the system specifications that Urban area includes "Paris" and "Peripheral road".

For the traditional mutants' scores, TS2_3Option2 killed 24 of the mutants alive for TS2_3Option1, and only one mutant killed by TS2_3Option1 was missed by TS2_3Option2: COR_3 (change of logical operator from && to || in *departureTime* ≥ 7 && *departureTime* < 24 for Sunday block of code), missed due to the uncovered scenario number 2 listed for TS2_3Option2 above.

In terms of comparison of DTs, Table 17 brings both resulting DTs and a summary of the issues left in each of them. The numbers of branches (leaves) are very close (eleven vs. 12) and the two main categories involved are the same: C 1 and C 2, with the addition of category C 5 at DT_{option2}. The numbers of misclassified ATCs are the same, six in different numbers of branches, and the CP elements not used in the DT have categories C 3 and C 4 in common, while in DT_{option1} C 5 was also unused, and in DT_{option2} the OEC EC 5 was unused. One could say that heuristic 2 results are slightly more efficient for pointing EC 5 as unused in DT_{option2}, but it is difficult to state that as the new test cases added to each ATS were distinct and the problems that may exist in one ATS may not exist in the other.

TS2_3Option1	TS2_3Option2
<p>J48 unpruned tree -----</p> <p>C 2 = Ch 2.8: EC 7 (3.0/1.0) M1 C 2 = Ch 2.4 C 1 = Ch 1.2: EC 6 (2.0/1.0) M2 C 1 = Ch 1.3: EC 7 (1.0) C 1 = Ch 1.1: EC 5 (2.0) C 1 = Ch 1.4: EC 6 (0.0) E1 C 2 = Ch 2.6: EC 4 (4.0/1.0) M3 C 2 = Ch 2.2: EC 7 (4.0/1.0) M4 C 2 = Ch 2.1: EC 4 (1.0) C 2 = Ch 2.5: EC 6 (3.0/2.0) M5 C 2 = Ch 2.3: EC 4 (5.0) C 2 = Ch 2.7: EC 4 (1.0)</p> <p>Number of Leaves : 11</p> <p>Size of the tree : 13</p>	<p>J48 unpruned tree -----</p> <p>C 1 = Ch 1.1: EC 6 (8.0/4.0) M1 C 1 = Ch 1.2 C 2 = Ch 2.4: EC 6 (2.0/1.0) M2 C 2 = Ch 2.6: EC 7 (1.0) C 2 = Ch 2.2: EC 7 (3.0/1.0) M3 C 2 = Ch 2.1: EC 4 (1.0) C 2 = Ch 2.8: EC 7 (1.0) C 2 = Ch 2.5: EC 6 (1.0) C 2 = Ch 2.3: EC 6 (1.0) C 2 = Ch 2.7: EC 6 (0.0) E1 C 1 = Ch 1.3 C 5 = Ch 5.2: EC 7 (6.0) C 5 = Ch 5.1: EC 4 (2.0) C 1 = Ch 1.4: EC 4 (1.0)</p> <p>Number of Leaves : 12</p> <p>Size of the tree : 15</p>
<p>six Misclassifications in five different branches. seven CP Elements not used in the DT: choices in C 3, C 4, and C 5.</p>	<p>six Misclassifications in three different branches. six CP Elements not used in the DT: choices in C 3, C4, and the OEC EC 5.</p>

Table 17 DTs Comparison

This experiment proves that when augmenting a test suite by adding TCs that correspond to diverse combinations of choices, the tree structure can indeed be affected by the set of choices combined in the new ATCs. This shows that the choices used in the combinations do matter as they may influence on the system's input-output relationships reflected in the tree, and mainly on the quality and effectiveness of the resulting TS. According to our results, it was not possible to state that one heuristic was better than the other one in terms of the resulting DT and possible problems left. On the other hand, in the comparison of code coverage percentages and mutation scores, there seem to be a tendency for having better results with the heuristic 2 (selecting choices with maximum count) in comparison to heuristic 1 (selecting choices with minimum count).

6. CONCLUSION

Melba is a method to be used in the context of black-box testing to help the test engineer in understanding a test suite, identifying needs for improvements and applying modifications, as well as validating their impact on the test suite effectiveness in relation to the specifications of the system under test. It is based on Machine Learning and Category-Partition (CP) testing, offers an iterative partially-automated process, and relies on the test engineer's knowledge of the system to provide the system's specifications and select the most suitable changes to the test suite.

This thesis extends previous work on Melba in several ways. First, it provides an implementation of the Melba process, with a user-friendly interface. Second, we expanded the method itself by (1) identifying new ways to detect problems and areas for improvements of a test suite and CP specification and (2) investigating and adopting standards and valuable additions to the analysis of possible problems. Third, the application of Melba in a case study not only investigates the results of our work but also validates and verifies our tool. The case study, thanks to the tool support, also allowed us to investigate what may impact the Melba process. These contributions were first introduced in [6].

The technology explored in order to automate the steps in the Melba process involved the use of external tools, such as TRex, Weka, and Janino, and the solutions implemented offer a tool that supports the user step by step in the use of the method and in the decision-making process when applying modifications to the specifications or to the test suite.

In the second aspect of our work, the method was improved with the adoption of the TTCN-3 standard format for the test suites, and the introduction of the possibility to define CP constraints (properties and selectors) that are now used, together with the tester's system definitions and the DT information, in the analysis of results, investigation of problems, and suggestion of new test cases. The method was also enhanced with three new analyses of observations and possible problems: (i) addition of the information that an output equivalence class could be ill-defined or useless just like a choice can be; (ii) verification of level of association between choices of different categories as a possible cause for CP elements to be unused in the DT; and (iii) distinction between unused

combinations of choices resulting from empty leaves in the DT and the ones suggested as missing in the structure of the tree.

As a third contribution, we have submitted the tool and the improved method to a case study that has been investigated in three distinct experiments: Does the Melba tool help improve the CP Specification; What is the impact the initial TS size *has on Melba's result when improving a TS?*; What is the impact the selection of choices has on the method *when adding TCs?* The case study was useful not only to study these three questions, but also to make some clarifications to the method. One clarification, mainly from Experiment 2 and Experiment 3, was that during the construction of the Decision Tree, there is no guarantee that all the input-output relationships are represented in the final DT since its structure depends on the test suite analyzed. The resulting DT and the improvements suggested by Melba indeed heavily depend on the initial test suite. Another related point is the great importance of the tester's knowledge in obtaining good results in the process. The tester should be an expert on the SUT's specifications in order to be able to identify problems in the rules present in the DT, and also to distinguish between faults that are due to problems in the CP specification and the ones that can lead to real improvements to the test suite. Experiment 1 has shown that a poor CP specification together with a good test suite can result in valuable improvements to the CP specification and even to the TS, as long as the tester has enough knowledge to evaluate each possible advice provided by Melba (which was also observed and described in the summary of Experiment 2). Experiment 2 has shown that a good CP specification together with a poor test suite can produce valid improvements to the test suite, but no guarantee is given that a minimum percentage neither of coverage nor of fault detection can be reached with the final TS. Future work will look into whether improvements to the Melba method (and tool) can address that issue. Experiment 3 has illustrated how the decisions on parameters characteristics (choices) for new test cases can influence on the DT structure and resulting test suite's structural coverage and fault effectiveness.

In this study we have also observed that both method and tool present possible limitations that could prevent the use of Melba on larger case studies. One issue could be the visualization and interpretation of larger Decision Trees as their analysis has shown to play a significant role while understanding and solving potential problems. A second

issue could be related to the definition of large CP Specifications, in which case it could probably be helpful to have ways of loading the specifications from other sources or formats. One last issue would be related to the types of software that are likely handled (and not handled) by the method and/or by the tool. An example would be systems with a state-based behaviour, i.e., a behaviour based on history. Melba would work as long as that history can be specified through Category-Partition.

6.1. Future Work

Based on the observations and conclusions of this work, some issues and investigations are still to be explored in the future:

- In the analysis of Case 1: Misclassifications (Section 3.4.1.1), the information about choices that are involved in multiple misclassifications seems not to be very helpful in solving the misclassifications, and for various situations in the experiments looking at the branches of the tree does not bring any intuition as to the root cause of misclassifications. Other studies need to investigate additional heuristics that could effectively support finding solutions to these cases;
- The current approach used for the Test of Association between two different categories (recall from Section 3.4.1.2) relies on the categories and choices that appear in the DT. This is a subset of the categories and choices involved in the ATS. Future work will look at whether using all these categories and choices (instead of the subset that appear in the DT) could give better results in the identification of strong associations. With the current method, the degree of association may end up being computed on a small sample, thus with a large possible error;
- Another point of improvement in the method is to try different procedures for the suggestion of new test cases as well as for the suggestion of redundant ones, e.g. by using the Base Choice approach;
- The Case Study has shown the need for a more extensive study on the sensitivity of the approach to its inputs (such as initial test suites as seen on Experiment 2);
- An enhancement for the method's results evaluation would be to investigate a different technique for the assessment of the final test suite. In this thesis, we

observed that the line and branch coverage of the test suite is usually only slightly affected by the elimination or addition of test cases. Other coverage criteria such as Data Flow Coverage could be used in future work.

In addition, some improvements are still to be done in future versions of the tool and here we list the main ones:

- The identification of instances of Case 4: Too Many Test Cases for a Rule, and to point out proper refinements for them, including the removal of test cases;
- The application of the adapter pattern in the communication between the Melba Tool and Weka in order to be able to easily use future releases of Weka, or to change the machine learning package;
- In the user interaction, Results interface, the “Analyzed” checkbox is currently volatile, i.e. the DT analysis is not persisted to make it possible to keep track of the “Analyzed” option once the Results interface is closed. The ideal will be to persist the DT analysis data and then, even if the tool is restarted, the user would still be able to continue on a previous unfinished analysis;
- Still for the Results interface, the tool may be improved to report differently on the actual and potential problems, and on the false positives. In the Experiments the reported Observations of Case 3: Missing Combinations of Choices presented a minimum of 82% of false positives, when not 100% as for most cases, and for the Observations of Case 5: Empty Leaves practically all cases presented 100% of false positives (except for a few cases where the combination was apparently feasible but actually unfeasible for the output class of that branch). A new approach to try and decrease the number of false positives could be experimenting with Pruned Decision Trees as, according to [35] (p.198), when reading rules from the tree branches they may be more complex than necessary, and this complexity could be reduced by pruning the rules based on the estimated error rate of each node. When nodes are eliminated, it becomes easier to identify duplicated rules that could be removed from the rule set, possibly reducing the number of false positives in our method;

- More flexibility on the TTCN-3 file should be offered by accepting more options from the language notation, and eliminating some of the current restrictions. Other test suite input file formats could also be supported;
- In the automatic execution and interaction with the DT generator algorithm, resources need to be provided for the tester to be able to customize some of the execution parameters, having the same freedom of execution as when running it directly from the generator itself (in our case Weka).
- The generation of test frames (cases) from the CP Specification.

All things considered, the main point of the current work is now having an automated tool that provides means to apply the method, now further improved, to larger case studies and to fully explore the promising future investigations.

REFERENCES

- [1] Ammann P. and Offutt A. J., *Introduction to Software Testing*, Cambridge University Press, 2008.
- [2] Andrews J. H., Briand L. C. and Labiche Y., "Is mutation an appropriate tool for testing experiments?," *Proc. the 27th ICSE*, 2005.
- [3] Bewick V., Cheek L. and Ball J., "Statistics review 8: Qualitative data – tests of association," in *Crit Care* 2004, vol. 8, pp. 46-53.
- [4] Bland J. M. and Altman D. G., "Statistics notes: Multiple significance tests: the Bonferroni method," *BMJ*, vol. 310 (6973), pp. 170-, 1995.
- [5] Briand L. C., Labiche Y. and Bawar Z., "Using Machine Learning to Refine Black-box Test Specifications and Test Suites," in *Carleton University: Technical Report SCE-07-05*, 2007.
- [6] Briand L. C., Labiche Y., Bawar Z. and Spido N. T., "Using machine learning to refine Category-Partition test specifications and test suites," *Information and Software Technology*, vol. 51 (11), pp. 1551-1564, 2009.
- [7] Daya S., "Yates' continuity correction," *Evidence-based Obstetrics and Gynaecology*, vol. 3 (4), pp. 169-169, 2001.
- [8] DeMillo R. A., Lipton R. J. and Sayward F. G., "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11 (4), pp. 34-41, 1978.
- [9] Doliner M., Cobertura, <http://cobertura.sourceforge.net/> (Last accessed July 2009)
- [10] England U. o. N., Bonferroni adjustment, http://www.une.edu.au/WebStat/unit_materials/c7_anova/oneway_bonferroni_adjust.htm (Last accessed May 2009)
- [11] Ernst M. D., Perkins J. H., Guo P. J., McCamant S., Pacheco C., Tschantz M. S. and Xiao C., "The Daikon system for dynamic detection of likely invariants," *Science of Computer Programming*, 2007.
- [12] Everitt B. S., *The analysis of contingency tables*, Chapman & Hall, 2 Edition, 1992.
- [13] Göttingen U. and SWE, TRex, the TTCN-3 Refactoring and Metrics Tool, <http://www.trex.informatik.uni-goettingen.de/trac/wiki/WikiStart> (Last accessed May 2009)
- [14] Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P. and Witten I. H., "The WEKA Data Mining Software: An Update," in *SIGKDD Explorations*, vol. 11, Issue 1, 2009.
- [15] Heim S. G., *The Resonant Interface*, Pearson - Addison Wesley, 2007.
- [16] Heim S. G., "The Resonant Interface - Chapter 6 Design Principles," Pearson - Addison Wesley, pp. 193-232, 2007.
- [17] Heim S. G., "The Resonant Interface - Chapter 10 Interface Components," Pearson - Addison Wesley, pp. 361-406, 2007.
- [18] Jeffrey D. and Gupta N., "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Transactions on Software Engineering*, vol. 33 (2), pp. 108-123, 2007.
- [19] L'Ecuyer P. and Montréal U. d., SSJ, Stochastic Simulation in Java, <http://www.iro.umontreal.ca/~simardr/ssj/indexe.html> (Last accessed May 2009)

- [20] Leitner A., Ciupa I., Oriol M., Meyer B. and Fiva A., "Contract driven development = test driven development - writing test cases," *Proc. the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007.
- [21] Liu S. and Chen Y., "A relation-based method combining functional and structural testing for test case generation," *The Journal of Systems & Software*, vol. 81 (2), pp. 234-248, 2008.
- [22] Ma Y. S., Offutt J. and Kwon Y. R., "MuJava: a mutation system for java," *Proc. the 28th ICSE*, 2006.
- [23] Mountainminds G. C. K. and Hoffmann M. R., EcEmma - Java Code Coverage for Eclipse, <http://www.eclemma.org/> (Last accessed November 2009)
- [24] Neukirchen H. and Bisanz M., "Utilising code smells to detect quality problems in TTCN-3 test suites," *Lecture Notes in Computer Science*, vol. 4581, pp. 228, 2007.
- [25] Offutt J., Ma Y. S. and Kwon Y. R., "An experimental mutation system for Java," *SIGSOFT Softw. Eng. Notes*, vol. 29 (5), pp. 1-4, 2004.
- [26] OMG, UTP - UML Testing Profile, http://www.omg.org/technology/documents/formal/test_profile.htm (Last accessed June 2009)
- [27] Ostrand T. J. and Balcer M. J., "The Category-Partition Method for Specifying and Generating Functional Test," *Communications of the ACM*, vol. 31 (6), pp. 676-686, 1988.
- [28] Quinlan J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.
- [29] Santelices R., Chittimalli P. K., Apiwattanapong T., Orso A. and Harrold M. J., "Test-Suite Augmentation for Evolving Software," *Proc. Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on*, pp. 218-227, 2008.
- [30] Simple, Simple - XML Serialization, <http://simple.sourceforge.net/> (Last accessed May 2009)
- [31] Tigris.org, Subversion, version control system, <http://subversion.tigris.org/> (Last accessed May 2009)
- [32] TMate S., SVNKit, [Sub]Versioning for Java, <http://svnkit.com/> (Last accessed May 2009)
- [33] Unkrig A., Janino, an Embedded Java Compiler, <http://www.janino.net/> (Last accessed May 2009)
- [34] Willcock C., Deiß D., Tobies S., Keil S., Engler F. and Schulz S., *An Introduction to TTCN-3*, Wiley, 2005.
- [35] Witten I. H. and Frank E., *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufman, 2nd Edition, 2005.
- [36] Wohlin C., Höst M., Runeson P., Ohlsson M. C., Regnell B. and Wesslén A., *Experimentation in software engineering: an introduction*, Kluwer Academic Pub, 2000.
- [37] Zhao L. and Elbaum S., "Quality assurance under the open source development model," *The Journal of Systems & Software*, vol. 66 (1), pp. 65-75, 2003.

APPENDIX A THE χ^2 (CHI-SQUARED) DISTRIBUTION METHOD

Here we assume that we are interested in evaluating the degree of association of two categories A and B. The seven steps of the method are detailed below, with a running example.

We use the example of Figure 53 as a running example. In the example of Figure 53 we have an ATS of 31 ATCs (31 instances classified in the DT). Among the categories defined for the CP Specification of that example, category C 8 was not used in the DT, and we are interested in checking its possible associations to the other categories that were used in the DT: C 1, C 3, C 5, and C 7. The degrees of association need to be measured separately for each one of them against C 8, and here we illustrate the process of verifying the association of C 8 with category C 1.

1. Computing the observed frequencies of ATCs in a contingency table

We first count the number of ATCs that exercise each choice of categories A and B. A contingency table summarizes the observed frequencies of ATCs for pairs of choices from A and B. Assuming A has i choices, A.1, A.2, ... A. i , and B has j choices, B.1, B.2, ... B. j , the contingency table will show in an $i \times j$ table how many ATCs exercise choices A. i and B. j at the same time.

On our running example, Figure 54 shows the contingency table for C 8 and C 1.

2. Continuity Correction for 2 x 2 contingency tables

As the chi-squared distribution in tests of association is an approximation that assumes reasonably large expected frequencies, when testing the relationship between two categories each with only two choices (the special case of 2 x 2 tables), a variation of the method is used: Yates' continuity correction [7] which is performed before applying chi-squared, in order to improve the approximation. Applying Yates' continuity correction to the frequencies means to make the test closer to the exact test and it is done by increasing the smallest observed frequency in the table by 0.5 and adjusting the other frequencies accordingly to keep the row and column totals the same. The chi-squared test then uses these adjusted values, instead of the original frequency values, and presents improved results for the specific case of 2 x 2 tables.

In our example, the continuity correction is not applied as the contingency table is rather a 2 x 4 table: 2 choices in C 8 against 4 choices in C 1.

Figure 53 Sample DT: used category C 1 and unused category C 8

C 8	C 1			
	Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4
	Ch 8.1	Ch 8.2	Ch 8.3	Ch 8.4
	1.0	6.0	0.0	1.0
	1.0	13.0	0.0	0.0

Figure 54 Observed Frequencies for categories C 8 and C 1

3. Calculating the expected frequencies for each element in the contingency table

The expected frequencies for each element in the contingency table are the frequencies of ATCs expected under the null hypothesis that there is no association between the categories. The null hypothesis means the proportions in each choice of one category would remain the same regardless of the choices of the other category. The expected frequency of each element of the contingency table is given by:

$$E_{ij} = T_{ChA,i} * \left(\frac{T_{ChB,j}}{N_{CB}} \right),$$

where E_{ij} is the expected frequency for row i , column j , $T_{ChA.i}$ is the total of ATCs for choice Ch A. i (total frequency for row i), $T_{ChB.j}$ is the total of ATCs for choice Ch B. j (total frequency for column j), and N_{CB} is the total frequency of ATCs classified under category C B branches. As the degree of association may vary across paths in the DT, we decided to consider only the ATCs under C B branches as opposed to considering all ATCs in the ATS, which could only provide an approximate picture of the association.

In our example, Figure 55 (a) shows the expected frequencies calculated for each element of the table, e.g., for table cell Ch 8.1 vs. Ch 1.1, the calculation was given by²⁰:

$$E_{ij} = 8 * \left(\frac{2}{22} \right) = 0.727$$

C 8	C 1			
	Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4
	Ch 8.1	Ch 8.2	Ch 8.3	Ch 8.4
	0.727	6.909	0.000	0.364
	1.273	12.091	0.000	0.636

(a)

C 8	C 1			
	Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4
	Ch 8.1	Ch 8.2	Ch 8.3	Ch 8.4
	0.102	0.120	0.000	1.114
	0.058	0.068	0.000	0.636

(b)

Figure 55 Expected frequencies (a) and test statistic values (b) for C 8 and C 1

4. Calculating the test statistics for each element of the contingency table

The test statistic for each element should be calculated as the difference between the observed and the expected frequencies, and scaled according to the expected frequency of that cell of the table, according to:

$$S_{ij} = \frac{(O_{ij} - E_{ij})^2}{E_{ij}},$$

where S_{ij} is the scaled test statistic for the element at row i , column j in the contingency table, O_{ij} is the observed frequency for row i , column j , and E_{ij} is the expected frequency for row i , column j .

Figure 55 (b) shows the table for the test statistics values of each element in our example. For instance, at table cell Ch 8.1 vs. Ch 1.1 the formula will give:

²⁰ It is important to note that the total of ATCs used are 22, instead of the total 31, due to considering here only the ones classified under the DT branches where C 1 is involved.

$$S_{ij} = \frac{(1 - 0.727)^2}{0.727} = 0.102$$

5. Calculating the chi-squared approximation sum

The chi-squared approximation sum is the total test statistics, $TotalS_{ij}$, for a table with r rows and c columns given by:

$$TotalS_{ij} = \sum_{i=1}^r \sum_{j=1}^c S_{ij}$$

In the example, the test statistic values of Figure 55 (b) are added resulting in the approximated total test statistics:

$$TotalS_{ij} = 0.102 + 0.120 + 0 + 1.114 + 0.058 + 0.068 + 0 + 0.636 = 2.099$$

6. Calculating the probability of no association P

P is the probability of obtaining frequencies like the ones observed, under the hypothesis that there is no association between categories C A and C B. If this hypothesis is true, the total test statistics, $TotalS_{ij}$, approximately follows an χ^2 distribution with $(r - 1) * (c - 1)$ degrees of freedom, and this approximation can be used to obtain a P value by looking at an χ^2 table [36], or by using a statistical package (e.g., the one presented in Section 4.4.5).

P should be evaluated according to a critical significance level (alpha) which is usually 5% (0.05). However, as the probability of getting a false significant result increases with the number of comparisons, Bonferroni adjustment [4, 10] was applied to the evaluation of P by dividing alpha by the number of comparisons of a specific category to other categories, and then only suggesting that the two analyzed categories are strongly associated when P value is less than this threshold of alpha divided by the number of comparisons. This makes the probability of no association P more trustful by being harder to claim a significant result.

Based on the value obtained in our example for the total test statistics, $TotalS_{ij} = 2.099$ with $(2 - 1) * (4 - 1) = 3$ degrees of freedom, a P value of 0.552179 is obtained from an χ^2 table. As C 8 is compared to the 4 categories in the DT, a threshold of $(0.05 / 4) = 0.0125$ is applicable, which is less than the value of P and therefore prevents the suggestion of any association between C 8 and C 1.

7. Calculating the residuals for each element in the contingency table

The chi-squared test, by means of P value, indicates whether there is an association between two categorical variables but it does not give any indication of the strength of the association. In order to do that, it is also necessary to identify the cells that have large differences between the observed and the expected frequencies. These differences are referred to as residuals, and here they are calculated as to be standardized and adjusted to follow a Normal distribution with mean 0 and standard deviation 1 [12], according to:

$$d_{ij} = \frac{O_{ij} - E_{ij}}{\sqrt{E_{ij} \left(1 - \frac{T_{ChA.i}}{N_{CB}}\right) \left(1 - \frac{T_{ChB.j}}{N_{CB}}\right)}}$$

where d_{ij} is the residual value for row i , column j , $T_{ChA.i}$ is the total of ATCs for choice Ch A. i , $T_{ChB.j}$ is the total of ATCs for choice Ch B. j , and N_{CB} is the total frequency of ATCs classified under C B branches. The larger the absolute value of the residual, the larger the difference between the observed and expected frequencies, and therefore the more significant the association between the two categories.

The table of Figure 56 brings the residual values computed for each element, e.g., for table cell Ch 8.1 vs. Ch 1.1, the residual is given by:

$$d_{11} = \frac{1 - 0.727}{\sqrt{0.727 \left(1 - \frac{8}{22}\right) \left(1 - \frac{2}{22}\right)}} = 0.4209$$

		C 1			
C 8		Ch 1.1	Ch 1.2	Ch 1.3	Ch 1.4
	Ch 8.1	0.420	-1.174	0.000	1.354
	Ch 8.2	-0.420	1.174	0.000	-1.354

Total test statistic (Chi Squared approximation): 2.099 with 3 degrees of freedom
Probability of no association* P = 0.552179

Figure 56 Residuals (table), total test statistic and P value for categories C 8 and C 1

By analyzing the pieces of information resulting from these 7 steps (Figure 56), together with similar results calculated for C 8 against each of the other categories in the DT, the tester may detect and correct undesirable associations of categories in the ATS, consequently applying improvements to the variability of the TS.

APPENDIX B VENDING MACHINE EXAMPLE

B.1. System Specification

System specification for a coin-based vending machine that issues drinks and food to customers:

- The price for all the items that are in a given machine is the same: \$1.00. Products available are: water, coca-cola, sprite, fanta, cookies, and chips.
- Users insert coins to purchase an item. Valid coins are: \$0.05, \$0.10, \$0.25, \$1.00, and \$2.00. Whenever pennies are inserted, no product should be released and the refund is automatically processed.
- Customers can have a refund of their money.
- The vending machine counts the amount of money inserted only when user demands an action, such as a refund of the money or the purchase of an item.
- The machine returns proper change in case the amount inserted is greater than \$1.00.

B.2. Category-partition Specification

For Input Parameters:

Parameter: **value_inserted**

Category 1 – possible values for insertion

*Ch 1.1: no coin inserted [error]

$\text{value_inserted} == 0$

Ch 1.2: penny inserted [error]

$\text{value_inserted} == 0.01$

Ch 1.3: nickel inserted

$\text{value_inserted} == 0.05$

Ch 1.4: dime inserted

$\text{value_inserted} == 0.10$

Ch 1.5: quarter inserted

$\text{value_inserted} == 0.25$

Ch 1.6: looney inserted

$\text{value_inserted} == 1$

Ch 1.7: twoney inserted

value_inserted == 2

Parameter: **item_wanted**

Category 2 - available items

*Ch 2.1: water

item_wanted == "water"

Ch 2.2: soft drink

item_wanted == "coca-cola" || item_wanted == "sprite" || item_wanted == "fanta"

Ch 2.3: cookies

item_wanted == "cookies"

Ch 2.4: chips

item_wanted == "chips"

Ch 2.5: refund

item_wanted == "refund"

For Output Parameters:

Parameter: **product_released**

Equivalence Class 1: drink released

product_released == "water" || product == "coca-cola" || product == "sprite" || product == "fanta"

Equivalence Class 2: food released

product_released == "cookies" || product == "chips"

Equivalence Class 3: no product released

product_released == ""

Parameter: **refund_value**

Equivalence Class 4: no value to refund

refund_value == 0

Equivalence Class 5: refund is the same as value inserted

refund_value == value_inserted

Equivalence Class 6: refund is the change - value inserted less the price of item purchased (\$1)

refund_value == value_inserted - 1

* base choice

APPENDIX C CONSTRAINTS ON THE CONCEPTUAL MODEL

Context AbstractTC:

// An abstract test case has as many choices as categories and only one choice for each category.

```
self.choice.category->asSet()->size() = self.choice->asSet()->size()
and
self.choice.category=Category.allInstances
```

Context CPSpec:

// The selectors and properties of the CPSpec are the ones used for choices

```
self.selectors = self.parameter.category.choice.selector->asSet()
and
self.properties = self.parameter.category.choice.property->asSet()
```

Context SystemUnderTest:

// The system under test has as many test cases as abstract test cases

```
self.testSuite.testCase->asSet()->size() = self.abstractTS.abstractTC->asSet()->size()
```

Context Category:

// The list of choices of a category includes the base choice

```
self.choices->includes(self.baseChoice)
```

Context Correlation:

// The (abstract) test cases involved in the correlation are the abstract test cases of the system under test

```
self.testCases = self.abstractTS.abstractTC->asSet()
```

Context AbstractTS:

// The categories involved in an abstract test suite are the ones involved in this test suite's abstract test cases

```
self.category = self.abstracTC.choice.category->asSet()
```

Context TestSuite:

```
// The parameters of the test suite (through records) are the parameters of the test cases
```

```
self.record.parameter->asSet() = self.testCase.paramValue.parameter->asSet()
```

Context Parameter:

```
// input parameters and environment variables are classified with categories whereas  
output parameters are classified with output equivalence classes
```

```
(self.type = ParamType::Param_Type_Input or self.type =
```

```
ParamType::Param_Type_Env_Var) and
```

```
self.classification.oclIsTypeOf(Category)
```

```
or
```

```
self.type = ParamType::Param_Type_Output and
```

```
self.classification.oclIsTypeOf(OutputEquivalenceClass)
```

Context DecisionTreeNode:

```
// A tree node is not its parent or sub-node, and its parent is not a sub-node
```

```
// and, if the node is linked to an analysis, it does not have a parent (it is the root)
```

```
not self.parent = self
```

```
and
```

```
not self.subNodes->includes(self)
```

```
and
```

```
not self.subNodes->includes(self.parent)
```

```
and
```

```
self.decisionTreeAnalysis->notEmpty implies self.parentNode->isEmpty
```

Context DecisionTreeLeaf:

```
// A leaf does not have sub-nodes and
```

```
// if the number of misclassified instances is not zero, then there is a misclassification
```

```
self.subNodes->isEmpty
```

and

```
Misclassification.allInstances->select(m|self.AbstractTC->includes(m.AbstractTC))  
->size = self.incorrectlyClassified
```

Context MissingChoiceCombination:

```
// A missing combination of choices is a list of choices that belong to different categories  
self.choicesCombination->forAll(c1,c2:Choice| not c1.category = c2.category)
```

APPENDIX D SAMPLE OF TTCN-3 TEST SUITE FILE

This file is a sample test suite for the PackHexChar program containing 2 test cases.

<pre> module PackHexChar { type record packHexCharRequest { integer rlen, integer odd_digit, charstring s } type record packHexCharResponse { integer v, integer returnValue } template packHexCharRequest request1 := { rlen := 0, odd_digit := -1, s := "A" } template packHexCharResponse response1 := { v := -1, returnValue := -1 } template packHexCharRequest request2 := { rlen := 0, odd_digit := 10, s := "A" } template packHexCharResponse response2 := { v := -1, returnValue := 10 } </pre>	<pre> type port packHexCharPort message { in packHexCharResponse; out packHexCharRequest; } type component MTCType { port packHexCharPort packHexCharOffice; } testcase test1() runs on MTCType { packHexCharOffice.send(request1); alt { [] } packHexCharOffice.receive(response1) { setverdict(pass) } [] packHexCharOffice.receive { setverdict(fail) } } testcase test2() runs on MTCType { packHexCharOffice.send(request2); alt { [] } packHexCharOffice.receive(response2) { setverdict(pass) } [] packHexCharOffice.receive { setverdict(fail) } } control { execute(test1()) execute(test2()) } </pre>
--	--

APPENDIX E Pre-Defined Operations for CP Specification

TTCN-3 supported types (Java data type ²¹)	Operations	
boolean (boolean)	1. true 2. false	
integer (int) and float (double)	1. equal to 6. different from 2. greater than 7. odd 3. less than 8. even 4. greater than or equal 9. inclusive interval 5. less than or equal 10. exclusive interval	
charstring (String)	1. equals 12. substring 2. equalsIgnoreCase 13. allHexadecimal 3. contains 14. nonHexadecimal 4. endsWith 15. mixedHexadecimal (!hexadecimal && !nonHexadecimal) 5. startsWith 16. numberOfHexaChars 6. indexOf 17. allNumbers 7. lastIndexOf 18. allUpperCase 8. empty 19. allLowerCase 9. null 20. mixedCase (!allUpperCase && !allLowerCase) 10. length 21. evenHexaChars 11. matches 22. oddHexaChars	

²¹ Java data type for the parameter when created in Melba tool after read from TTCN-3 TS file.

APPENDIX F THE TAXI BILLING SYSTEM

F.1. Initial defective Category-partition Specification for Case Study - Experiment 1, following Melba Tool format

For Input Parameters:

Parameter: **destination**

Category 1 - Area of destination

*Ch 1.1: Urban destination

`destination.equals("Paris") || destination.equals("Peripheral road")`

Ch 1.2: Suburban destination

`destination.equals("End of area") || destination.equals("Orly") ||`

`destination.equals("Roissy") || destination.equals("Du Parc") ||`

`destination.equals("Villepinte")`

Ch 1.3: Beyond Suburban destination

`destination.equals("Beyond suburban")`

Ch 1.4: Destination not informed or unknown [error]

`destination.isEmpty() || (!destination.equals("Paris") &&`

`!destination.equals("Peripheral road") && !destination.equals("End of area")`

`&&`

`!destination.equals("Orly") && !destination.equals("Roissy") &&`

`!destination.equals("Du Parc") && !destination.equals("Villepinte") &&`

`!destination.equals("Beyond suburban"))`

Parameter: **departureTime**

Category 2 - Time of departure

Ch 2.1: Negative time [error]

`departureTime < 0`

Ch 2.2: Night

`departureTime >= 0 && departureTime < 7`

Ch 2.3: Morning

`departureTime >= 7 && departureTime < 10`

*Ch 2.4: Afternoon

departureTime >= 10 && departureTime < 17

Ch 2.5: Early evening

departureTime >= 17 && departureTime < 19

Ch 2.6: Late evening

departureTime >= 19 && departureTime < 24

Ch 2.7: Illegal time [error]

departureTime >= 24

Ch 2.8: Sundays special time of departure

departureTime >= 7 && departureTime < 24

Parameter: **dayOfWeek**

Category 3 - Day of the week

Ch 3.1: Day not informed or unknown [error]

dayOfWeek.isEmpty() || (!dayOfWeek.equals("Monday") &&
!dayOfWeek.equals("Tuesday") && !dayOfWeek.equals("Wednesday") &&
!dayOfWeek.equals("Thursday") && !dayOfWeek.equals("Friday") &&
!dayOfWeek.equals("Saturday") && !dayOfWeek.equals("Sunday"))

*Ch 3.2: Weekdays and Saturdays

dayOfWeek.equals("Monday") || dayOfWeek.equals("Tuesday") ||
dayOfWeek.equals("Wednesday") || dayOfWeek.equals("Thursday") ||
dayOfWeek.equals("Friday") || dayOfWeek.equals("Saturday")

Ch 3.3: Sundays

dayOfWeek.equals("Sunday")

Parameter: **holiday**

Category 4 - Holiday

Ch 4.1: It is a holiday

true

*Ch 4.2: It is not a holiday

false

Parameter: **kmRun**

Category 5 - Kilometres run

Ch 5.1: Illegal distance run [error]

kmRun <= 0

*Ch 5.2: Legal distance run

kmRun > 0

For Output Parameters:

Parameter: **totalCost**

Equivalence Class 1: Total cost was less than minimum cost established

totalCost == 5.80

Equivalence Class 2: Total cost is greater than minimum cost established

totalCost >= 5.80

Equivalence Class 3: Error in input data, total could not be calculated

totalCost < 0

Equivalence Class 8: Total cost is between 0 and 5.80

(0 < totalCost) && (totalCost < 5.80)

Parameter: **chargeApplied**

Equivalence Class 4: Error in input data, charge was not used

isEmpty

Equivalence Class 5: Charge A

chargeApplied.equals("A")

Equivalence Class 6: Charge B

chargeApplied.equals("B")

Equivalence Class 7: Charge C

chargeApplied.equals("C")

* base choice

F.2. Initial well-structured Test Suite for Case Study - Experiment 1

<i>Taxi Billing System</i>							
Test Cases	Input Parameters					Output Parameters	
	destination	Departure Time	dayOfWeek	holiday	kmRun	totalCost	chargeApplied
TC1	"Paris"	0	"Monday"	false	1	5.80	"B"
TC2	"Paris"	7	"Tuesday"	false	2	5.80	"B"
TC3	"Paris"	10	"Wednesday"	false	3	5.80	"A"

TC4	"Paris"	17	"Thursday"	false	4	6.68	"B"
TC5	"Paris"	19	"Friday"	false	5	7.80	"B"
TC6	"Paris"	0	"Monday"	true	6	8.92	"B"
TC7	"Paris"	7	"Saturday"	false	7	10.04	"B"
TC8	"Paris"	0	"Sunday"	false	7	11.65	"C"
TC9	"Paris"	0	"Sunday"	true	6	10.3	"C"
TC10	"Paris"	7	"Sunday"	false	5	7.8	"B"
TC11	"Paris"	7	"Sunday"	true	4	6.68	"B"
TC12	"Orly"	0	"Saturday"	false	1	5.80	"C"
TC13	"Orly"	7	"Friday"	false	2	5.80	"B"
TC14	"Orly"	10	"Thursday"	false	3	5.80	"B"
TC15	"Orly"	17	"Wednesday"	false	4	6.68	"B"
TC16	"Orly"	19	"Tuesday"	false	5	8.95	"C"
TC17	"Orly"	0	"Monday"	true	6	10.3	"C"
TC18	"Orly"	0	"Sunday"	false	7	11.65	"C"
TC19	"Orly"	0	"Sunday"	true	6	10.3	"C"
TC20	"Orly"	7	"Sunday"	false	5	8.95	"C"
TC21	"Orly"	7	"Sunday"	true	4	7.6	"C"
TC22	"Beyond suburban"	0	"Saturday"	false	7	11.65	"C"
TC23	"Beyond suburban"	7	"Monday"	false	6	10.3	"C"
TC24	"Beyond suburban"	10	"Tuesday"	false	5	8.95	"C"
TC25	"Beyond suburban"	17	"Friday"	false	1	5.80	"C"
TC26	"Beyond suburban"	19	"Wednesday"	false	3	6.25	"C"
TC27	"Beyond suburban"	0	"Thursday"	false	2	5.80	"C"
TC28	"Beyond suburban"	7	"Saturday"	true	4	7.6	"C"
TC29	"Beyond suburban"	0	"Sunday"	true	2	5.80	"C"
TC30	"Beyond suburban"	0	"Sunday"	false	3	6.25	"C"
TC31	"Beyond suburban"	7	"Sunday"	true	5	8.95	"C"
TC32	"Beyond suburban"	7	"Sunday"	false	9	14.35	"C"
TC33	"	7	"Tuesday"	false	9	-1	"
TC34	"Unknown"	7	"Tuesday"	true	9	-1	"
TC35	"Beyond suburban"	-1	"Sunday"	false	9	-1	"
TC36	"Orly"	-2	"Sunday"	true	9	-1	"
TC37	"Orly"	25	"Sunday"	true	9	-1	"
TC38	"Paris"	100	"Tuesday"	true	9	-1	"
TC39	"Orly"	100	"Tuesday"	false	9	-1	"
TC40	"Beyond suburban"	0	"Tuesday"	false	0	-1	"
TC41	"Beyond suburban"	7	"Tuesday"	false	0	-1	"
TC42	"Beyond suburban"	10	"Tuesday"	false	0	-1	"
TC43	"Beyond suburban"	17	"Tuesday"	false	0	-1	"
TC44	"Beyond suburban"	19	"Tuesday"	false	0	-1	"
TC45	"Paris"	0	"Wednesday"	false	-1	-1	"
TC46	"Paris"	7	"Wednesday"	false	-1	-1	"
TC47	"Paris"	10	"Wednesday"	false	-1	-1	"
TC48	"Paris"	17	"Wednesday"	false	-1	-1	"
TC49	"Paris"	19	"Wednesday"	false	-1	-1	"
TC50	"Orly"	0	"Saturday"	false	-10	-1	"
TC51	"Orly"	7	"Saturday"	false	-10	-1	"
TC52	"Orly"	10	"Saturday"	false	-10	-1	"
TC53	"Orly"	17	"Saturday"	false	-10	-1	"
TC54	"Orly"	19	"Saturday"	false	-10	-1	"
TC55	"Orly"	0	"	true	5	-1	"
TC56	"Paris"	0	"Unknown"	true	5	-1	"

TC57	"Paris"	10	"Wednesday"	true	6	8.92	"B"
TC58	"Paris"	16	"Wednesday"	false	7	8.22	"A"
TC59	"Paris"	16	"Wednesday"	true	6	8.92	"B"
TC60	"Orly"	7	"Friday"	true	5	8.95	"C"
TC61	"Orly"	10	"Thursday"	true	9	14.35	"C"
TC62	"Orly"	17	"Wednesday"	true	4	7.60	"C"
TC63	"Orly"	9	"Friday"	true	5	8.95	"C"
TC64	"Orly"	16	"Thursday"	true	9	14.35	"C"
TC65	"Orly"	18	"Wednesday"	true	4	7.6	"C"
TC66	"Orly"	9	"Friday"	false	2	5.80	"B"
TC67	"Orly"	16	"Thursday"	false	3	5.80	"B"
TC68	"Orly"	18	"Wednesday"	false	4	6.68	"B"
TC69	"Roissy"	20	"Thursday"	true	2	5.80	"C"
TC70	"Peripheral road"	8	"Monday"	true	7	10.04	"B"
TC71	"Paris"	18	"Tuesday"	true	8	11.16	"B"
TC72	"Peripheral road"	23	"Friday"	true	10	13.40	"B"

F.3. Category-partition Specification resulting from Case Study - Experiment 1, following Melba Tool format

For Input Parameters:

Parameter: **destination**

Category 1 - Area of destination

*Ch 1.1: Urban destination

destination.equals("Paris") || destination.equals("Peripheral road")

Ch 1.2: Suburban destination

destination.equals("End of area") || destination.equals("Orly") ||

destination.equals("Roissy") || destination.equals("Du Parc") ||

destination.equals("Villepinte")

Ch 1.3: Beyond Suburban destination

destination.equals("Beyond suburban")

Ch 1.4: Destination not informed or unknown [error]

destination.isEmpty() || (!destination.equals("Paris") &&

!destination.equals("Peripheral road") && !destination.equals("End of area")

&&

!destination.equals("Orly") && !destination.equals("Roissy") &&

!destination.equals("Du Parc") && !destination.equals("Villepinte") &&

!destination.equals("Beyond suburban"))

Parameter: **departureTime**

Category 2 - Time of departure

Ch 2.1: Negative time [error]

departureTime < 0

Ch 2.2: Night

departureTime >= 0 && departureTime < 7

Ch 2.3: Morning [if !Sunday]

departureTime >= 7 && departureTime < 10 &&
!dayOfWeek.equals("Sunday")

*Ch 2.4: Afternoon [if !Sunday]

departureTime >= 10 && departureTime < 17 &&
!dayOfWeek.equals("Sunday")

Ch 2.5: Early evening [if !Sunday]

departureTime >= 17 && departureTime < 19 &&
!dayOfWeek.equals("Sunday")

Ch 2.6: Late evening [if !Sunday]

departureTime >= 19 && departureTime < 24 &&
!dayOfWeek.equals("Sunday")

Ch 2.7: Illegal time [error]

departureTime >= 24

Ch 2.8: Sundays special time of departure [if !Weekday]

departureTime >= 7 && departureTime < 24 && dayOfWeek.equals("Sunday")

Parameter: **dayOfWeek**

Category 3 - Day of the week

Ch 3.1: Day not informed or unknown [error]

dayOfWeek.isEmpty() || (!dayOfWeek.equals("Monday") &&
!dayOfWeek.equals("Tuesday") && !dayOfWeek.equals("Wednesday") &&
!dayOfWeek.equals("Thursday") && !dayOfWeek.equals("Friday") &&
!dayOfWeek.equals("Saturday") && !dayOfWeek.equals("Sunday"))

*Ch 3.2: Weekdays and Saturdays [properties Weekday]

dayOfWeek.equals("Monday") || dayOfWeek.equals("Tuesday") ||
dayOfWeek.equals("Wednesday") || dayOfWeek.equals("Thursday") ||

`dayOfWeek.equals("Friday") || dayOfWeek.equals("Saturday")`

Ch 3.3: Sundays [properties Sunday]

`dayOfWeek.equals("Sunday")`

Parameter: **holiday**

Category 4 - Holiday

Ch 4.1: It is a holiday

`true`

*Ch 4.2: It is not a holiday

`false`

Parameter: **kmRun**

Category 5 - Kilometres run

Ch 5.1: Illegal distance run [error]

`kmRun <= 0`

*Ch 5.2: Legal distance run

`kmRun > 0`

For Output Parameters:

Parameter: **totalCost**

Equivalence Class 1: Total cost is less than minimum cost established

`totalCost < 5.80`

Equivalence Class 2: Total cost is greater than or equal to minimum cost established

`totalCost >= 5.80`

Parameter: **chargeApplied**

Equivalence Class 4: Error in input data, charge was not used

`isEmpty`

Equivalence Class 5: Charge A

`chargeApplied.equals("A")`

Equivalence Class 6: Charge B

`chargeApplied.equals("B")`

Equivalence Class 7: Charge C


```
chargeApplied.equals("C")
```

```
* base choice
```

F.4. Application of alternative solution for Case Study Experiment 1 – Execution 1

At Iteration 2, looking closer at the Misclassifications reported, we perform here the experimentation of trying to redefine the CP Specification in order to eliminate the 5 Misclassifications spotted. Since the OECs initially defined for totalCost are totally unrelated to the chargeApplied, being totally dependent on the kmRun, we need some input definition that relates the chargeApplied to the kmRun. Thus, the following change is applied to category C 5:

```
Parameter: kmRun
```

```
Category 5 - Kilometres run
```

```
Ch 5.1: Illegal distance run      [error]
```

```
kmRun <= 0
```

```
*Ch 5.2: Distance run is in interval for charge C and minimum cost
```

```
kmRun > 0 && kmRun <= 2.66
```

```
Ch 5.3: Distance run is in interval for charge B and minimum cost
```

```
kmRun > 2.66 && kmRun <= 3.21
```

```
Ch 5.4: Distance run is in interval for charge A and minimum cost
```

```
kmRun > 3.21 && kmRun <= 4.18
```

```
Ch 5.5: Distance run results in cost greater than minimum
```

```
kmRun > 4.18
```

Alternative Iteration 3

With category C 5 changed, we now obtain the reduced DT of Figure 57 with size 25 and 20 leaves, and 121 Nonconformities were identified, showing that the 5 Misclassifications were indeed eliminated. The 3 unused elements correspond to the two choices in category C 4 that were used in the ATS but not in the tree, and the OEC EC 8, not used in the ATS (and actually useless as analyzed in the main steps of the experiment – Section 5.2.2, Iteration 2). The category C 4 is identified as possibly strongly associated to C 5.(probability of no association $P = 0.000348$), therefore new test cases could be added to eliminate that association. Other than that, the Empty Leaf reported is an

unfeasible combination of choices for involving Error choices Ch 1.4. And last, from the 117 unused combinations of choices, 21 are suggested as feasible, however several of them are not actually feasible as they suggest choices Ch 2.3 to Ch 2.6 (departure times for weekday) combined with Ch 3.3 (day of week Sunday), and as a consequence corrections similar to the ones applied at the main steps of the experiment (Section 5.2.2, Iteration 2) would be applicable here before adding the new test cases.

Figure 57 DT for Alternative Iteration 3 with category C 5 changed

This experiment partial solution was about showing that multiple solutions can be applied as corrections to the problems identified by Melba. It is up to the tester to analyze what is the best approach to correct them, according to the knowledge of the system's specification, and improve the CP Specification so that the method can produce a better DT and spot additional possibilities for improvements of the TS.

F.5. Test Suite TS2_1 for Case Study

Taxi Billing System							
Test Cases	Input Parameters					Output Parameters	
	destination	Departure Time	dayOfWeek	holiday	kmRun	totalCost	chargeApplied
TC1	"Paris"	0	"Monday"	false	1	5.80	"B"

TC2	"Paris"	7	"Tuesday"	false	2	5.80	"B"
TC3	"Paris"	10	"Wednesday"	false	3	5.80	"A"
TC4	"Paris"	17	"Thursday"	false	4	6.68	"B"
TC5	"Paris"	19	"Friday"	false	5	7.80	"B"
TC6	"Paris"	0	"Monday"	true	6	8.92	"B"
TC7	"Paris"	7	"Saturday"	false	7	10.04	"B"
TC8	"Paris"	0	"Sunday"	false	7	11.65	"C"
TC9	"Paris"	0	"Sunday"	true	6	10.3	"C"
TC10	"Paris"	7	"Sunday"	false	5	7.8	"B"
TC11	"Paris"	7	"Sunday"	true	4	6.68	"B"
TC12	"Orly"	0	"Saturday"	false	1	5.80	"C"
TC13	"End of area"	7	"Friday"	false	2	5.80	"B"
TC14	"Orly"	10	"Thursday"	false	3	5.80	"B"
TC15	"Du Parc"	17	"Wednesday"	false	4	6.68	"B"
TC16	"Orly"	19	"Tuesday"	false	5	8.95	"C"
TC17	"Villepinte"	0	"Monday"	true	6	10.3	"C"
TC18	"Orly"	0	"Sunday"	false	7	11.65	"C"
TC19	"Orly"	0	"Sunday"	true	6	10.3	"C"
TC20	"Orly"	7	"Sunday"	false	5	8.95	"C"
TC21	"Orly"	7	"Sunday"	true	4	7.6	"C"
TC22	"Beyond suburban"	0	"Saturday"	false	7	11.65	"C"
TC23	"Beyond suburban"	7	"Monday"	false	6	10.3	"C"
TC24	"Beyond suburban"	10	"Tuesday"	false	5	8.95	"C"
TC25	"Beyond suburban"	17	"Friday"	false	1	5.80	"C"
TC26	"Beyond suburban"	19	"Wednesday"	false	3	6.25	"C"
TC27	"Beyond suburban"	0	"Thursday"	false	2	5.80	"C"
TC28	"Beyond suburban"	7	"Saturday"	true	4	7.6	"C"
TC29	"Beyond suburban"	0	"Sunday"	true	2	5.80	"C"
TC30	"Beyond suburban"	0	"Sunday"	false	3	6.25	"C"
TC31	"Beyond suburban"	7	"Sunday"	true	5	8.95	"C"
TC32	"Beyond suburban"	7	"Sunday"	false	9	14.35	"C"
TC33	"	7	"Tuesday"	false	9	-1	"
TC34	"Unknown"	7	"Tuesday"	true	9	-1	"
TC35	"Beyond suburban"	-1	"Sunday"	false	9	-1	"
TC36	"Orly"	-2	"Sunday"	true	9	-1	"
TC37	"Orly"	25	"Sunday"	true	9	-1	"
TC38	"Paris"	100	"Tuesday"	true	9	-1	"
TC39	"Orly"	100	"Tuesday"	false	9	-1	"
TC40	"Beyond suburban"	0	"Tuesday"	true	0	-1	"
TC41	"Beyond suburban"	7	"Tuesday"	true	0	-1	"
TC42	"Beyond suburban"	10	"Tuesday"	true	0	-1	"
TC43	"Beyond suburban"	17	"Tuesday"	false	0	-1	"
TC44	"Beyond suburban"	19	"Tuesday"	false	0	-1	"
TC45	"Paris"	0	"Wednesday"	false	-1	-1	"
TC46	"Paris"	7	"Wednesday"	false	-1	-1	"
TC47	"Paris"	10	"Wednesday"	false	-1	-1	"
TC48	"Paris"	17	"Wednesday"	false	-1	-1	"
TC49	"Paris"	19	"Wednesday"	false	-1	-1	"
TC50	"Orly"	0	"Saturday"	true	-10	-1	"
TC51	"Orly"	7	"Saturday"	true	-10	-1	"
TC52	"Orly"	10	"Saturday"	true	-10	-1	"
TC53	"Orly"	17	"Saturday"	true	-10	-1	"
TC54	"Orly"	19	"Saturday"	true	-10	-1	"

TC55	"Orly"	0	"	true	5	-1	"
TC56	"Paris"	0	"Unknown"	true	5	-1	"
TC57	"Paris"	10	"Wednesday"	true	6	8.92	"B"
TC58	"Paris"	16	"Wednesday"	false	7	8.22	"A"
TC59	"Paris"	16	"Wednesday"	true	6	8.92	"B"
TC60	"Orly"	7	"Friday"	true	5	8.95	"C"
TC61	"Orly"	10	"Thursday"	true	9	14.35	"C"
TC62	"Orly"	17	"Wednesday"	true	4	7.60	"C"
TC63	"Orly"	9	"Friday"	true	5	8.95	"C"
TC64	"Orly"	16	"Thursday"	true	9	14.35	"C"
TC65	"Orly"	18	"Wednesday"	true	4	7.6	"C"
TC66	"Orly"	9	"Friday"	false	2	5.80	"B"
TC67	"Orly"	16	"Thursday"	false	3	5.80	"B"
TC68	"Orly"	18	"Wednesday"	false	4	6.68	"B"
TC69	"Roissy"	20	"Thursday"	true	2	5.80	"C"
TC70	"Peripheral road"	8	"Monday"	true	7	10.04	"B"
TC71	"Paris"	18	"Tuesday"	true	8	11.16	"B"
TC72	"Peripheral road"	23	"Friday"	true	10	13.40	"B"
TC73	"Beyond suburban"	16	"Tuesday"	true	10	15.70	"C"

F.6. TCs removed from TS2_1 resulting in TS2_2

The following Test Cases numbers were removed from TS2_1 in order to produce TS2_2: 1, 5-7, 9-10, 12, 15, 18, 20-23, 26-29, 37, 39-40, 43, 45-47, 49, 51, 53-57, 59-60, 62-63, 66-68, 70-73.

F.7. TCs removed from TS2_2 resulting in TS2_3

The following Test Cases numbers were removed from TS2_2 in order to produce TS2_3: 2, 13, 19, 32-33, 35-36, 42, 48, 52, 61, 65, 69.

F.8. Test Cases and Abstract Test Cases added to the Test Suite TS2_2, resulting in TS2_2Melba - Case Study

Input Parameters					Output Parameters	
destination	departureTime	dayOfWeek	holiday	kmRun	totalCost	chargeApplied
test1 Scenario2						
"Peripheral road" Ch 1.1	18 Ch 2.5	" Ch 3.1	true Ch 4.1	2 Ch 5.2	-1 EC 1	" EC 4
test2 Scenario2						
"Villepinte" Ch 1.2	9 Ch 2.8	"Sunday" Ch 3.3	false Ch 4.2	5 Ch 5.2	8.95 EC 2	"C" EC 7
test3 Scenario2						
"Beyond suburban" Ch 1.3	8 Ch 2.3	"Friday" Ch 3.2	false Ch 4.2	7 Ch 5.2	11.65 EC 2	"C" EC 7
test4 Scenario2						
"Beyond suburban"	20	"Tuesday"	true	10	15.70	"C"

Ch 1.3	Ch 2.6	Ch 3.2	Ch 4.1	Ch 5.2	EC 2	EC 7
test5 Scenario2						
"Paris" Ch 1.1	23 Ch 2.6	"Thursday" Ch 3.2	false Ch 4.2	6 Ch 5.2	8.92 EC 2	"B" EC 6
test6 Scenario2						
"Paris" Ch 1.1	-1 Ch 2.1	" Ch 3.1	true Ch 4.1	4 Ch 5.2	-1 EC 1	" EC 4
test7 Scenario2						
"Peripheral road" Ch 1.1	-2 Ch 2.1	"Unknown" Ch 3.1	true Ch 4.1	0 Ch 5.1	-1 EC 1	" EC 4
test8 Scenario2						
"Paris" Ch 1.1	-3 Ch 2.1	"Thursday" Ch 3.2	true Ch 4.1	6 Ch 5.2	-1 EC 1	" EC 4
test9 Scenario2						
"Orly" Ch 1.2	-4 Ch 2.1	"Thursday" Ch 3.2	false Ch 4.2	6 Ch 5.2	-1 EC 1	" EC 4
test10 Scenario2						
"End of area" Ch 1.2	6 Ch 2.2	" Ch 3.1	false Ch 4.2	-4 Ch 5.1	-1 EC 1	" EC 4
test11 Scenario2						
"Roissy" Ch 1.2	5 Ch 2.2	" Ch 3.1	false Ch 4.2	-9 Ch 5.1	-1 EC 1	" EC 4
test12 Scenario2						
"Villepinte" Ch 1.2	7 Ch 2.3	"Unknown" Ch 3.1	false Ch 4.2	-1 Ch 5.1	-1 EC 1	" EC 4
test13 Scenario2						
"Beyond suburban" Ch 1.3	8 Ch 2.3	"Unknown" Ch 3.1	true Ch 4.1	0 Ch 5.1	-1 EC 1	" EC 4
test14 Scenario2						
" Ch 1.4	10 Ch 2.4	" Ch 3.1	true Ch 4.1	11 Ch 5.2	-1 EC 1	" EC 4
test15 Scenario2						
"Beyond suburban" Ch 1.3	23 Ch 2.6	" Ch 3.1	true Ch 4.1	2 Ch 5.2	-1 EC 1	" EC 4
test16 Scenario2						
"Paris" Ch 1.1	24 Ch 2.7	" Ch 3.1	true Ch 4.1	5 Ch 5.2	-1 EC 1	" EC 4
test17 Scenario2						
"Du Parc" Ch 1.2	27 Ch 2.7	"Sunday" Ch 3.3	false Ch 4.2	7 Ch 5.2	-1 EC 1	" EC 4
test18 Scenario2						
"Beyond suburban" Ch 1.3	10 Ch 2.8	"Unknown" Ch 3.1	true Ch 4.1	8 Ch 5.2	-1 EC 1	" EC 4
test19 Scenario2						
"Paris" Ch 1.1	10 Ch 2.4	"Monday" Ch 3.2	true Ch 4.1	8 Ch 5.2	11.16 EC 2	"B" EC 6
test20 Scenario2						
"Peripheral road" Ch 1.1	10 Ch 2.4	"Monday" Ch 3.2	false Ch 4.2	8 Ch 5.2	9.08 EC 2	"A" EC 5
test21 Scenario2						
"Paris" Ch 1.1	11 Ch 2.4	"Wednesday" Ch 3.2	true Ch 4.1	12 Ch 5.2	15.64 EC 2	"B" EC 6
test22 Scenario2						
"Beyond suburban"	5	"Friday"	false	15	22.45	"C"

Ch 1.3	Ch 2.2	Ch 3.2	Ch 4.2	Ch 5.2	EC 2	EC 7
test23 Scenario2						
"Paris" Ch 1.1	9 Ch 2.3	"Saturday" Ch 3.2	true Ch 4.1	10 Ch 5.2	13.40 EC 2	"B" EC 6
test24 Scenario2						
"Peripheral road" Ch 1.1	18 Ch 2.5	"Saturday" Ch 3.2	true Ch 4.1	7 Ch 5.2	10.04 EC 2	"B" EC 6
test25 Scenario2						
"Paris" Ch 1.1	22 Ch 2.6	"Thursday" Ch 3.2	true Ch 4.1	3 Ch 5.2	5.80 EC 2	"B" EC 6
test26 Scenario2						
"Peripheral road" Ch 1.1	4 Ch 2.2	"Wednesday" Ch 3.2	true Ch 4.1	20 Ch 5.2	24.60 EC 2	"B" EC 6
test27 Scenario2						
"Paris" Ch 1.1	0 Ch 2.2	"Tuesday" Ch 3.2	false Ch 4.2	2 Ch 5.2	5.80 EC 2	"B" EC 6

F.9. Test Cases added to the Test Suite TS2_3, resulting in TS2_3Melba – Case Study

test1 ts2 3						
"Paris" Ch 1.1	-1 Ch 2.1	"" Ch 3.1	false Ch 4.2	2 Ch 5.2	-1 EC 1	"" EC 4
test2 ts2 3						
"End of area" Ch 1.2	23 Ch 2.8	"Sunday" Ch 3.3	false Ch 4.2	2 Ch 5.2	5.80 EC 2	"C" EC 7
test3 ts2 3						
"Paris" Ch 1.1	7 Ch 2.3	"Tuesday" Ch 3.2	true Ch 4.1	5 Ch 5.2	7.80 EC 2	"B" EC 6
test4 ts2 3						
"Peripheral road" Ch 1.1	19 Ch 2.6	"Monday" Ch 3.2	true Ch 4.1	6 Ch 5.2	8.92 EC 2	"B" EC 6
test5 ts2 3						
"Villepinte" Ch 1.2	-2 Ch 2.1	"Wednesday" Ch 3.2	true Ch 4.1	4 Ch 5.2	-1 EC 1	"" EC 4
test6 ts2 3						
"Orly" Ch 1.2	8 Ch 2.3	"Thursday" Ch 3.2	false Ch 4.2	10 Ch 5.2	13.40 EC 2	"B" EC 6
test7 ts2 3						
"Roissy" Ch 1.2	18 Ch 2.5	"Friday" Ch 3.2	true Ch 4.1	1 Ch 5.2	5.80 EC 2	"C" EC 7
test8 ts2 3						
"Du Parc" Ch 1.2	25 Ch 2.7	"Saturday" Ch 3.2	false Ch 4.2	7 Ch 5.2	-1 EC 1	"" EC 4
test9 ts2 3						
"Beyond suburban" Ch 1.3	-24 Ch 2.1	"Tuesday" Ch 3.2	true Ch 4.1	12 Ch 5.2	-1 EC 1	"" EC 4
test10 ts2 3						
"Beyond suburban" Ch 1.3	30 Ch 2.7	"Wednesday" Ch 3.2	true Ch 4.1	31 Ch 5.2	-1 EC 1	"" EC 4
test11 ts2 3						
"Paris" Ch 1.1	Ch 2.1	"Sunday" Ch 3.3	true Ch 4.1	0 Ch 5.1	-1 EC 1	"" EC 4

test12 ts2 3						
"Villepinte" Ch 1.2	Ch 2.2	" Ch 3.1	true Ch 4.1	6 Ch 5.2	-1 EC 1	" EC 4
test13 ts2 3						
"Beyond suburban" Ch 1.3	Ch 2.3	" Ch 3.1	true Ch 4.1	0 Ch 5.1	-1 EC 1	" EC 4
test14 ts2 3						
"Peripheral road" Ch 1.1	Ch 2.4	" Ch 3.1	true Ch 4.1	10 Ch 5.2	-1 EC 1	" EC 4
test15 ts2 3						
"Roissy" Ch 1.2	Ch 2.5	" Ch 3.1	false Ch 4.2	0 Ch 5.1	-1 EC 1	" EC 4
test16 ts2 3						
"Beyond suburban" Ch 1.3	Ch 2.6	" Ch 3.1	false Ch 4.2	7 Ch 5.2	-1 EC 1	" EC 4
test17 ts2 3						
" Ch 1.4	Ch 2.7	" Ch 3.1	false Ch 4.2	0 Ch 5.1	-1 EC 1	" EC 4
test18 ts2 3						
" Ch 1.4	Ch 2.7	"Sunday" Ch 3.3	false Ch 4.2	31 Ch 5.2	-1 EC 1	" EC 4
test19 ts2 3						
"Paris" Ch 1.1	10 Ch 2.4	"Wednesday" Ch 3.2	true Ch 4.1	3 Ch 5.2	5.80 EC 2	"B" EC 6
test20 ts2 3						
"Paris"	16	"Wednesday"	true	7	10.04	"B"
test21 ts2 3						
"Orly"	10	"Thursday"	true	3	6.25	"C"
test22 ts2 3						
"Orly" Ch 1.2	8 Ch 2.3	"Thursday" Ch 3.2	true Ch 4.1	10 Ch 5.2	15.70 EC 2	"C" EC 7
test23 ts2 3						
"Orly"	16	"Thursday"	false	9	12.28	"B"
test24 ts2 3						
"Roissy"	18	"Friday"	false	1	5.80	"B"
test25 ts2 3						
"Peripheral road"	15	"Sunday"	false	8	11.16	"B"
test26 ts2 3						
"Paris"	9	"Tuesday"	false	5	7.8	"B"
test27 ts2 3						
"Paris"	17	"Thursday"	true	7	10.04	"B"
test28 ts2 3						
"Peripheral road"	22	"Friday"	false	4	6.68	"B"
test29 ts2 3						
"Paris"	6	"Sunday"	true	3	6.25	"C"
test30 ts2 3						
"End of area"	5	"Wednesday"	false	2	5.80	"C"
test31 ts2 3						
"Orly"	20	"Thursday"	true	3	6.25	"C"
test32 ts2 3						
"Roissy"	12	"Sunday"	true	6	10.3	"C"
test33 ts2 3						
"Orly"	8	" Ch 3.1	true	10	-1	" EC 4
test34 ts2 3						

"Paris"	9	"	false	5	-1	"
test35 ts2 3						
"Paris"	7	"	true	5	-1	"
test36 ts2 3						
"Orly"	8	"	false	10	-1	"
test37 ts2 3						
"Orly"	16	"	false	9	-1	"
test38 ts2 3						
"Paris"	16	"	true	7	-1	"
test39 ts2 3						
"Orly"	10	"	true	3	-1	"
test40 ts2 3						
"Beyond suburban"	10	"	false	5	-1	"
test41 ts2 3						
"Paris"	10	"	false	3	-1	"
test42 ts2 3						
"Roissy"	18	"	false	1	-1	"
test43 ts2 3						
"Beyond suburban"	17	"	false	1	-1	"
test44 ts2 3						
"Paris"	17	"	true	7.0	-1	"
test45 ts2 3						
"Paris"	17	"	false	4	-1	"
test46 ts2 3						
"Roissy"	18	"	true	1	-1	"
test47 ts2 3						
"Orly"	19	"	false	5	-1	"
test48 ts2 3						
"Peripheral road"	22	"	false	4	-1	"
test49 ts2 3						
"Orly"	20	"	true	3	-1	"
test50 ts2 3						
"Peripheral road"	19	"	true	6	-1	"
test51 ts2 3						
"Paris"	7	"Sunday"	true	0	-1	"
test52 ts2 3						
"Paris"	10	"Wednesday"	true	0	-1	"
test53 ts2 3						
"Peripheral road"	15	"Sunday"	false	0	-1	"
test54 ts2 3						
"Paris"	9	"Tuesday"	false	0	-1	"
test55 ts2 3						
"Paris"	17	"Thursday"	true	0	-1	"
test56 ts2 3						
"Peripheral road"	22	"Friday"	false	0	-1	"
test57 ts2 3						
"Paris"	6	"Sunday"	true	0	-1	"
test58 ts2 3						
"Paris"	16	"Wednesday"	false	0	-1	"
test59 ts2 3						
"Paris"	7	"Tuesday"	true	0	-1	"
test60 ts2 3						
"Paris"	17	"Thursday"	false	0	-1	"

test61 ts2 3						
"Peripheral road"	19	"Monday"	true	0	-1	"C"
test62 ts2 3						
"Paris"	0	"Sunday"	false	0	-1	"C"
test63 ts2 3						
"Orly"	10	"Thursday"	false	0	-1	"C"
test64 ts2 3						
"Orly"	19	"Tuesday"	false	0	-1	"C"
test65 ts2 3						
"Villepinte"	0	"Monday"	true	0	-1	"C"
test66 ts2 3						
"Orly"	10	"Thursday"	true	0	-1	"C"
test67 ts2 3						
"Orly"	8	"Thursday"	true	0	-1	"C"
test68 ts2 3						
"Roissy"	18	"Friday"	false	0	-1	"C"
test69 ts2 3						
"End of area"	23	"Sunday"	false	0	-1	"C"
test70 ts2 3						
"End of area"	5	"Wednesday"	false	0	-1	"C"
test71 ts2 3						
"Orly"	20	"Thursday"	true	0	-1	"C"
test72 ts2 3						
"Roissy"	12	"Sunday"	true	0	-1	"C"
test73 ts2 3						
"Orly"	8	"Thursday"	false	0	-1	"C"
test74 ts2 3						
"Roissy"	18	"Friday"	true	0	-1	"C"
test75 ts2 3						
"Beyond suburban"	10	"Tuesday"	false	0	-1	"C"
test76 ts2 3						
"Beyond suburban"	17	"Friday"	false	0	-1	"C"
test77 ts2 3						
"Beyond suburban"	0	"Sunday"	false	0	-1	"C"
test78 ts2 3						
"Beyond suburban"	7	"Sunday"	true	0	-1	"C"
test79 ts2 3						
"Villepinte"	5	"Sunday"	false	7	11.65	"C"
test80 ts2 3						
"Beyond suburban"	11	"Monday"	true	5	8.95	"C"
test81 ts2 3						
"Beyond suburban"	21	"Thursday"	false	8	13.00	"C"
test82 ts2 3						
"Beyond suburban"	4	"Monday"	true	9	14.35	"C"
test83 ts2 3						
"Beyond suburban"	9	"Tuesday"	false	4	7.60	"C"
test84 ts2 3						
"Paris"	0	"Wednesday"	true	4	6.68	"B"
test85 ts2 3						
"Peripheral road"	6	"Friday"	false	7	10.04	"B"
test86 ts2 3						
"Roissy"	2	"Sunday"	false	2	5.80	"C"

F.10. Test Suite TS2_3 - Case Study

Taxi Billing System							
Test Cases	Input Parameters					Output Parameters	
	destination	departure Time	dayOfWeek	holiday	kmRun	totalCost	chargeApplied
TC3	"Paris"	10	"Wednesday"	false	3	5.80	"A"
TC4	"Paris"	17	"Thursday"	false	4	6.68	"B"
TC8	"Paris"	0	"Sunday"	false	7	11.65	"C"
TC11	"Paris"	7	"Sunday"	true	4	6.68	"B"
TC14	"Orly"	10	"Thursday"	false	3	5.80	"B"
TC16	"Orly"	19	"Tuesday"	false	5	8.95	"C"
TC17	"Villepinte"	0	"Monday"	true	6	10.3	"C"
TC24	"Beyond suburban"	10	"Tuesday"	false	5	8.95	"C"
TC25	"Beyond suburban"	17	"Friday"	false	1	5.80	"C"
TC30	"Beyond suburban"	0	"Sunday"	false	3	6.25	"C"
TC31	"Beyond suburban"	7	"Sunday"	true	5	8.95	"C"
TC34	"Unknown"	7	"Tuesday"	true	9	-1	"C"
TC38	"Paris"	100	"Tuesday"	true	9	-1	"C"
TC41	"Beyond suburban"	7	"Tuesday"	true	0	-1	"C"
TC44	"Beyond suburban"	19	"Tuesday"	false	0	-1	"C"
TC50	"Orly"	0	"Saturday"	true	-10	-1	"C"
TC58	"Paris"	16	"Wednesday"	false	7	8.22	"A"
TC64	"Orly"	16	"Thursday"	true	9	14.35	"C"

F.11. Test Cases Added to TS2_3 at Experiment 3

Input Parameters					Output Parameters	
destination	departureTime	dayOfWeek	holiday	kmRun	totalCost	chargeApplied
===== Option 1 =====						
test1 option1						
"End of area"	-2	"Sunday"	true	4	-1	"C"
test2 option1						
"Orly"	8	"Sunday"	false	7	11.65	"C"
test3 option1						
"Beyond suburban"	22	"C"	true	5	-1	"C"
test4 option1						
"Paris"	20	"C"	true	2	-1	"C"
test5 option1						
"Orly"	17	"C"	true	10	-1	"C"
test6 option1						
"Roissy"	9	"C"	false	1	-1	"C"
test7 option1						
"Beyond suburban"	8	"C"	true	6	-1	"C"
test8 option1						
"Peripheral road"	7	"C"	true	12	-1	"C"
===== Option 2 =====						
test1 option2						
"Villepinte"	4	"Sunday"	false	11	17.05	"C"

test2 option2						
“Roissy”	-3	“”	false	13	-1	“”
test3 option2						
“Orly”	8	“Sunday”	false	7	11.65	“C”
test4 option2						
“Beyond suburban”	22	“Monday”	false	5	8.95	“C”
test5 option2						
“Paris”	20	“Tuesday”	false	2	5.80	“B”
test6 option2						
“Orly”	17	“Wednesday”	false	10	13.40	“B”
test7 option2						
“End of area”	9	“Thursday”	false	1	5.80	“B”
test8 option2						
“Beyond suburban”	8	“Saturday”	false	6	10.30	“C”
test9 option2						
“Peripheral road”	7	“Friday”	false	12	15.64	“B”